Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

PROYECTO FINAL DE CARRERA

TÍTULO DEL PFC: "Construcción de un sistema de gestión de descargas de imágenes Sentinel basado en tecnología web"

TITULACIÓN: Ingeniería de Telecomunicaciones (Segundo ciclo)

AUTOR: Angel Leonardo Guamán Sanginés Uriarte

DIRECTOR: Pablo Royo Chic

DATA: 6 de Febrero de 2017

Título: "Construcción de un sistema de gestión de descargas de imágenes Sentinel basado en tecnología web"

Autor: Angel Leonardo Guamán Sanginés Uriarte

Director: Pablo Royo Chic

Data: 6 de Febrero del 2017

Resumen

El presente documento describe los detalles para el diseño e implementación de un prototipo de una aplicación web que genere línea de costa (línea en la superficie de la Tierra que define el límite entre el mar y la tierra firme) a partir de imágenes satelitales Sentinel-2.

El proyecto se realizará en el "*Centre Tecnològic Telecomunicacions Catalunya*" (CTTC) en la división de Geomática.

Title: "Design & implementation of a web-based system for the management of the download of Sentinel imagery"

Author: Angel Leonardo Guamán Sanginés Uriarte

Director: Pablo Royo Chic

Date: February 6th 2017

Overview

This document describes the design and implementation of a web-based application prototype that generates shoreline charts (line that defines the boundary between Earth surface and the sea and the mainland) from Sentinel-2 satellite imagery.

The project will be developed at the "C*enter Tecnològic Telecomunicacions Catalunya*" (CTTC) in the Geomatics division.

Agradecimientos:

A mi familia, amigos y a mi tutor del CTTC por la a ayuda y paciencia otorgada.

ÍNDICE

INTRODUCCIÓN

El presente Proyecto Final de Carrera tiene por objeto el diseño y la implementación de un prototipo de una aplicación web que genere línea de costa (línea en la superficie de la Tierra que define el límite entre el mar y la tierra firme) a partir de imágenes satelitales de la plataforma Sentinel-2[1].

El proyecto se realizará en el "*Centre Tecnològic Telecomunicacions Catalunya*" (CTTC) [2] en la división de Geomática. El CTTC es una institución de investigación sin fines de lucro, dependiente del Gobierno Catalán, de la *Universitat Politècnica de Catalunya* y de la *Universidad Ramón Lllul*.

Actualmente no existe ningún sistema de producción de línea de costa en el CTTC. Experimentalmente la manera en que se procesan las imágenes de Sentinel-2 es de forma manual utilizando scripts en cada una de las etapas de generación de línea de costa. Cada etapa del proceso puede demorar varias horas, dependiendo, de la disponibilidad y de la capacidad de proceso del hardware que ejecute el procesado de la imagen. Además los procesos no se encuentran concatenados, por lo que es necesario que un operador controle e inicie cada una de sus etapas, si ocurre un error, no es posible detectarlo a tiempo y esto ocasiona el aumento del tiempo requerido para la generación de la línea de costa.

De este modo, el sistema expuesto en el presente documento permite realizar búsquedas de imágenes Sentinel-2 y su respectivo procesado para la generación de línea de costa, a través de una interfaz de usuario muy simple, en la cual, el usuario tendrá información actualizada de estado de cada proceso y a su finalización exitosa, la descarga local de la imagen solicitada con la línea de costa generada sobrepuesta. Además el sistema genera notificaciones cuando existe algún tipo de error.

CAPÍTULO 1. INTRODUCCIÓN Y CONTEXTO DEL PROYECTO

1.1. Estructura de la memoria

En este documento se hará un repaso de todos los elementos y procesos que han sido necesarios para la construcción de la aplicación. Desde la explicación del contexto y motivos para su desarrollo, sus requerimientos hasta entrar en el detalle de su arquitectura diseño e implementación.

La memoria del proyecto está estructurada en cuatro capítulos:

- En el capítulo primero se comentará brevemente las carencias actuales de la cartografía de línea de costa, luego se señalarán los objetivos del presente proyecto final de carrera, después describiremos los requerimientos del proyecto y al finalizar el capítulo se hará una escueta descripción de los satélites Sentinel.

- En el segundo capítulo, se efectuará una descripción de las etapas necesarias para obtener una línea de costa a partir de la plataforma web que posee Sentinel, para definir luego los elementos del sistema que permitirá detallar su arquitectura y diseño.

- El capítulo tercero describirá el entorno de desarrollo utilizado, se listarán las aplicaciones y las librerías que han sido requeridas y se describirá la implementación de cada elemento descrito en la arquitectura del proyecto.

- Para terminar, en el capítulo cuarto se exponen las conclusiones.

1.2.- Presentación y ámbito del proyecto

1.2.1.- Carencias en la cartografía de línea de costa

El Instituto de Recursos Mundial WRI [3] (*World Resources Institute*) reporta una longitud de costas total en el mundo de 1.634.000 km. Una parte importante de esas líneas de costa ha sido trazada cartográficamente. Sin embargo, existen aún muchos lugares en donde la calidad cartográfica existente es aún baja para realizar su trazado, o simplemente es inexistente. Esto sucede usualmente en áreas lejanas de las rutas de navegación comercial, pero también lugares no tan lejanos, incluso existen áreas en Europa afectadas por este problema, además nuevas áreas inexploradas están apareciendo en el Ártico debido a la fusión de la capa hielo polar.

Las grandes flotas de buques mercantes hacen uso de cartografía oficial actualizada pero, existen muchas pequeñas embarcaciones que no tienen acceso a la información actualizada necesaria para navegar con seguridad por zonas en donde no existe cartografía confiable. Las necesidades de este último grupo necesitan resolverse y ser satisfechas.

De este modo, con el fin de mitigar las carencias de la cartografía de línea de costa, el CTTC trabaja en el desarrollo de un sistema para la generación de información cartográfica, fiable y actualizada en base a la información proporcionada por los satélites Sentinel.

1.2.2.- Requerimientos

Para la generación de la línea de costa, como punto de partida, el usuario debe introducir a través de una interfaz gráfica (UI) las coordenadas (latitud, longitud) de la porción de terreno, luego el sistema debe buscar las imágenes en el servidor de Sentinel-2 y desplegarlos en una lista.

Una vez desplegados, la interfaz debe permitir al usuario seleccionar uno o más productos para la generación de línea de costa. El sistema debe ser capaz de generar automáticamente la (s) línea (s) de costa y notificar al usuario su estado. En caso de que se produzca algún error, el sistema también lo debe notificar al usuario.

Todo este proceso debe ser accesible a través de la interfaz de usuario.

1.3.- Objetivos del Proyecto Final de Carrera

1.3.1.- Objetivo General

El presente Proyecto Final de Carrera tiene por objeto el diseño y la implementación de un prototipo de sistema automatizado de producción de cartografía de costa que comprende desde la solicitud por parte del usuario hasta la entrega del producto, gestionándose automáticamente las diferentes fases necesarias para su consecución, respetando los límites de capacidad de la infraestructura disponible y generando como resultado final, una imagen en formato JPG con la línea de costa sobrepuesta.

1.3.2.- Objetivos específicos

Para cumplir con el objetivo general es necesario describir los siguientes objetivos específicos:

- Diseño e implementación de una interfaz de usuario
- Implementación de un servidor web
- Diseño e implementación de API para tratar peticiones de la interfaz de usuario.
- Diseño e implementación de un proceso informático de gestión de tareas
- Diseño e implementación de un modelo de datos
- Implementación de un proceso de generación de línea de costa
- Implementación de los procesos de :

    - Descarga de imágenes desde los servidores de Sentinel.
    - Descompresión de las imágenes descargadas.
    - Conversión de formato de las imágenes descomprimidas.

- Implementación de un sistema de notificaciones a través de correo electrónico.

El proyecto se convierte en el punto de partida de un sistema que a mediano-largo plazo será implementado por el CTTC para cubrir diversas exigencias cartográficas; que aparte de la generación de línea de costa, ofrecerá también un conjunto de aplicativos adicionales, para generar información cartográfica de fácil acceso y actualizada a partir de imágenes Sentinel.

CAPÍTULO 2. ANÁLISIS Y DISEÑO

2.1. Introducción

Para obtener la línea de costa de una porción de terreno, una vez que el usuario ha seleccionado sus coordenadas (latitud, longitud) el sistema identificará de forma transparente la imagen o imágenes disponibles dentro de los servidores de Sentinel, ofreciéndosele la posibilidad de escoger las que desee.

Una vez el usuario ha elegido la porción de terreno a procesar (creación de una tarea) el sistema automáticamente se encargará de ejecutar cada una de las etapas necesarias para generar la línea de costa y la entregará al usuario a través de un enlace de descarga. Todo el proceso podrá ser monitorizado desde una interfaz gráfica, además cuando el proceso haya finalizado, el sistema será capaz de notificar al usuario que la tarea ha sido ejecutada con éxito o que ha ocurrido un error.

Es este capítulo se detallarán aspectos del análisis y diseño del sistema a partir de los requerimientos y las condiciones impuestas por el CTTC.

En primer lugar se explicará brevemente dónde están disponibles las imágenes de los satélites Sentinel-2, luego se realizará una descripción de las etapas necesarias para la generación de la línea de costa y para finalizar, se describirá el diseño detallado del sistema.

2.2.- Imágenes de los Satélites Sentinel-2

Las imágenes Sentinel-2, están disponibles a los usuarios de forma gratuita a través de *“The Sentinels Scientific Data Hub”* que es un punto de acceso libre y gratuito para la descarga de los “productos” Sentinel-1 y Sentinel-2.

El *“Data Hub”* dispone de una atractiva interface de usuario *“Scientific Hub”* que proporciona herramientas de búsqueda y filtrado de fácil manejo así como también de una API denominada *“API Hub”* que está destinada a usuarios que utilizan scripts para automatizar la descarga de los “productos” Sentinel.

En este contexto, “producto” es un conjunto de datos (imágenes y otros tipos de datos) que contienen una compilación de porciones de terreno elementales de tamaño fijo.

En adelante se hará referencia a “producto”, como las imágenes proporcionadas por *“The Sentinels Scientific Data Hub”.*

2.3.- Etapas para la generación de la línea de costa

Antes de describir el diseño del sistema, es importante comprender las etapas necesarias para la generación de la línea de costa:

- Búsqueda de las imágenes Sentinel-2
- Descarga de las imágenes
- Descompresión de las imágenes
- Extracción y conversión de formato de las imágenes
- Extracción de línea de costa
- Compresión de las imágenes con línea de costa

2.3.1.- Búsqueda de las imágenes en la API de Sentinel

La búsqueda de las imágenes de los satélites Sentinel-2 se las realiza a través de la API de *"The Sentinels Scientific Data Hub"*, de acceso libre previo registro. El "*Data Hub"* ofrece una serie de herramientas para la descarga de las imágenes a través de URIs que permiten realizar consultas específicas, entrega respuestas contenidas en ficheros XML.

2.3.2.- Descarga y descompresión de las imágenes

Las imágenes que proporciona la API de Sentinel se descargan a través de un enlace web, están comprimidas en formato "zip", suelen tener un tamaño entre 1 a 10 gigabytes. La descarga es un proceso delicado que debe ser gestionada persistentemente.

Cada descarga contiene imágenes en formato JPG2000 correspondientes a varias secciones de terreno de 500 m^2 denominadas gránulos en distintas bandas espectrales (ver Anexo 1).

2.3.3.- Extracción y conversión de formato de las imágenes

Una vez las imágenes se han descargado y descomprimido correctamente el sistema debe seleccionar las imágenes correspondientes a las bandeas B03, B08 y B11 (Ver Anexo 1). Estas imágenes además, utilizan el formato de compresión JPG2000 y deben ser trasformadas al estándar JPG.

2.3.4.- Extracción de línea de costa

La línea de costa se extrae a partir de un algoritmo diseñado e implementado por el CTTC, que utiliza como datos de entrada imágenes de formato JPG correspondientes a las bandas B03, B08 y B11. Como salida genera una imagen de tipo JPG de una sección de terreno que contiene sobrepuesta al plano original la línea de costa generada.

2.3.5.- Compresión de las imágenes con línea de costa.

Cada imagen Sentinel descargada, es un conjunto de imágenes de varias porciones de terreno de 500 m^2 y de distintas bandas. El resultado final del proceso, son varias imágenes en formato JPG con la línea de costa generada. Para su respectiva entrega, el sistema debe comprimir estas imágenes y generar un solo fichero que se encontrará disponible a través de un link.

2.5.- Elementos del sistema

Es así que de acuerdo a los requerimientos y las estas etapas descritas en los párrafos anteriores se ha visto que el diseño del sistema debe estar compuesto por los siguientes elementos:

- Interfaz de usuario
- Servidor web
- Demonio TCP (proceso que encadena las etapas descritas en seccion 2.3)
- Base de datos
- Procesos

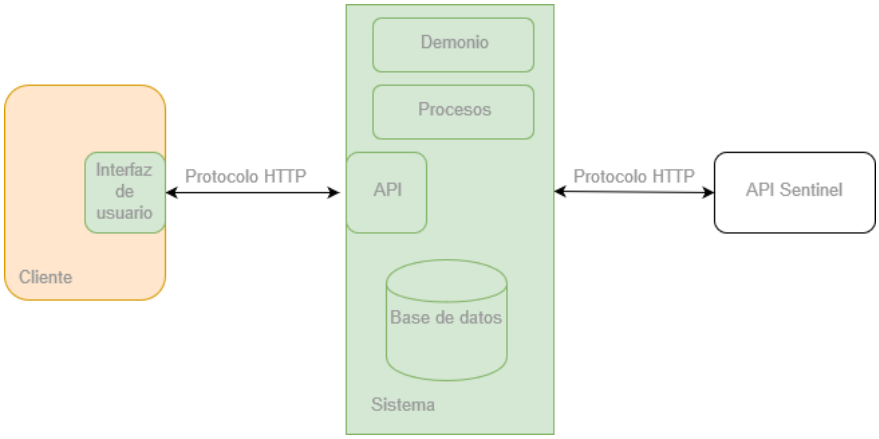


Figura 1: Esquema global del sistema, en color verde la parte diseñada e implementada

2.5.1.- Interfaz de Usuario

La interfaz de usuario es la parte del sistema visible, está implementada a través de una aplicación web, en ella se realizan las siguientes acciones:

- Introducción de coordenadas para la búsqueda: latitud, longitud.
- Despliegue de imágenes correspondientes a la búsqueda.
- Selección de imágenes a generar línea de costa.
- Monitorización de los procesos de generación de línea de costa.
- Descarga de las imágenes con línea de costa generadas.
- Visualización y modificación de información de la cuenta e información. personal

2.5.2.- Servidor web

El servidor web permite la implementación de una API que permite atender las peticiones generadas por la interfaz de usuario, así como también realiza consultas "*The Sentinels Scientific Data Hub",* procesa sus respuestas, crea tareas nuevas y finalmente notifica al demonio la creación de una tarea.

Una API cuyas siglas vienen del inglés "*Application Programming Interface"*, es una capa de abstracción, es un conjunto de funciones y procedimientos que cumplen una o muchas funciones con el fin de ser utilizadas por otro software.

Una tarea es un encargo por pare de un usuario para generar la línea de costa de un conjunto de imágenes Sentinel. Las tareas pasan por distintas etapas y el usuario puede tener N tareas.

2.5.3.- Demonio TCP

Un demonio o *daemon* (de sus siglas en inglés *Disk And Execution Monitor*), es un tipo especial de proceso informático no interactivo, que se ejecuta en segundo plano de forma continua.

El demonio es la parte fundamental del sistema porque cada que sucede un evento en el sistema, crea una lista de tareas a ejecutar según su prioridad y lanza uno a uno los procesos en una secuencia debida. Además controla la ejecución de los procesos lanzados.

2.5.4.- Base de datos

El diseño del modelo se implementa sobre una base de datos relacional, que a través de un proceso de mapeo es posible almacenar los objetos en tablas relacionales.

Cada que se produzca un evento, el demonio leerá la base de datos y creará una lista de tareas que deben ser atendidas, posteriormente a medida que cada tarea vaya ejecutándose, el demonio y los procesos actualizarán su estado.

2.5.5.- Procesos

Los procesos son las fases que, sucesivamente, conducen al producto final y que, ejecutados en el debido orden, sirven para convertir las imágenes de entrada en cartografía de línea de costa.

A continuación se muestra una tabla con la lista de los procesos utilizados y su orden de ejecución en relación a las estepas definidas en la sección 2.4.4.

ORDEN	PROCESO	¿DISEÑADO?	¿IMPLEMENTADO?
1	Descarga	Sí	Sí
2	Descompresión	No	Sí
3	Extracción de imágenes	Sí	Sí
4	Conversión de formato	No	Sí
5	Generación de línea de costa	No	Si
6	Compresión	No	Sí
7	Notificación		

Tabla 1: Lista de procesos utilizados en el sistema

2.6.- Arquitectura

2.6.1.- Descripción general

El sistema de generación de línea de costa es un conjunto de elementos, que están relacionados por el esquema de la Figura 2. Donde de color naranja señala la aplicación web, en verde se presentan los elementos diseñados e implementados en QT/C++, en azul la base de datos, en gris los elementos que son programas ya implementados y finalmente en azul claro se señala el elemento externo: La API de Sentinel.

La interfaz de usuario es una aplicación web escrita JavaScript + HTML, para su implementación se ha utilizado el framework AngularJS y tiene como función capturar las peticiones del usuario, transmitir éstas al servidor web así como informar al primero sobre las diferentes respuestas que el sistema tenga que notificar.

El servidor web permite la comunicación entre el sistema y la interfaz de usuario y/u otra futura aplicación. El servidor web es un servicio que se ejecuta constantemente una vez arranque el sistema operativo y sirve peticiones HTTP en un puerto determinado. Además este módulo se conectará con la API de Sentinel para realizar la búsqueda de las imágenes y adaptar su contenido para entregarlo a la interfaz de usuario.

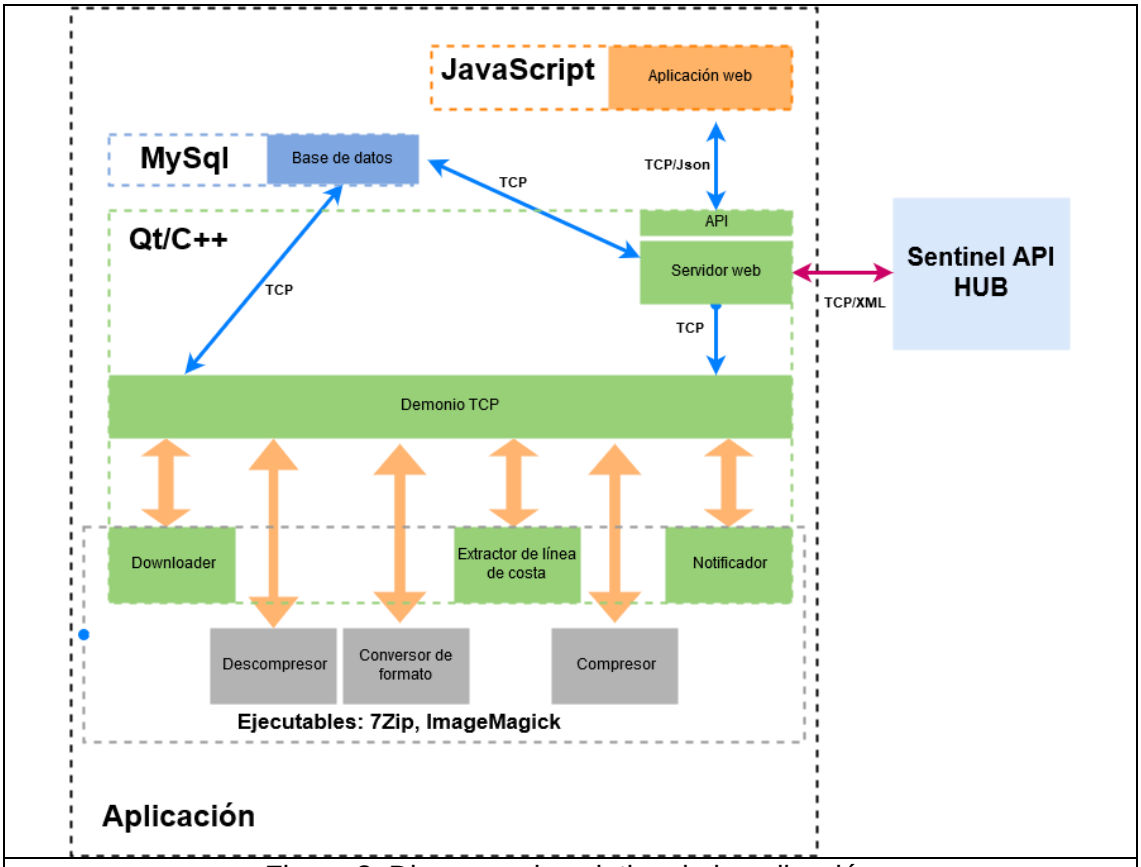


Figura 2: Diagrama descriptivo de la aplicación

Las peticiones que hace la interfaz web llegan a la API del sistema y son interpretadas. Una vez interpretadas, el servidor web procesa la petición y entrega una respuesta a la interfaz web.

Las respuestas que entrega el servidor son en la mayoría de los casos estructuras JSON que contienen datos solicitados o simplemente confirmaciones de acciones.

El servidor web lanza consultas contra la API de Sentinel cuando la petición que recibe es una búsqueda de imágenes. La plataforma Sentinel devuelve un listado de imágenes correspondientes a la búsqueda en formato XML que el servidor convierte a JSON y lo entrega a la interfaz web.

Cuando la interfaz solicita la generación de una línea de costa, envía una petición a la API del sistema con los parámetros requeridos, el servidor web crea una nueva tarea y envía una notificación al demonio a través de una conexión TCP de un solo sentido.

Una vez el demonio recibe la notificación, realiza una lectura de la base de datos y genera una lista de tareas a realizar. Cada tarea se sirve a través el lanzamiento de un proceso.

El Demonio TCP se encarga en general de la inteligencia de negocio, específicamente tiene las siguientes funciones:

- Gestionar los estados del sistema
- Lanzar procesos
- Gestionar notificaciones
- Actualizar la base de datos

El orden en que se lanzan los procesos es el que se indica en la Tabla 1, además, el demonio cada que lanza un proceso actualiza la base de datos en función si el lanzamiento ha sido exitoso o no.

Finalmente si todos los procesos se han ejecutado correctamente en un correcto orden el demonio envía un email al usuario con un enlace de descarga del producto final. Por otro lado, si sucede un error, el demonio también enviará un correo electrónico indicando que ha habido un error de tarea determinada.

En este contexto definimos producto final al conjunto de imágenes en donde cada una de ellas se corresponde a una sección de terreno de $500m^2$ relacionados con la longitud y latitud que el usuario introdujo en la interfaz y que contienen las líneas de costa solicitadas.

2.7.- Diseño

2.7.1.- Aplicación Web

La aplicación web representa la parte visible del sistema en su conjunto, convirtiéndose en la interfaz de usuario y básicamente tiene como objetivo capturar las peticiones de usuario a través de elementos HTML, realizar peticiones a la API del sistema en cuestión y desplegar los datos que esta le proporciona. De este modo, su diseño y usabilidad debe tener principal atención debido a que es la parte visible del presente Proyecto Final de Carrera.

Como se verá en la Sección 3.31, para su implementación se ha utilizado el framework AngularJS versión 1, y sus detalles se describirán en dicha sección.

La interfaz de usuario ha sido diseñada con un solo actor (Usuario), dejando el actor de Administrador para una futura implementación. En la Figura 3a se indican los casos de uso de la aplicación web, su descripción se encuentra en el Anexo 2.

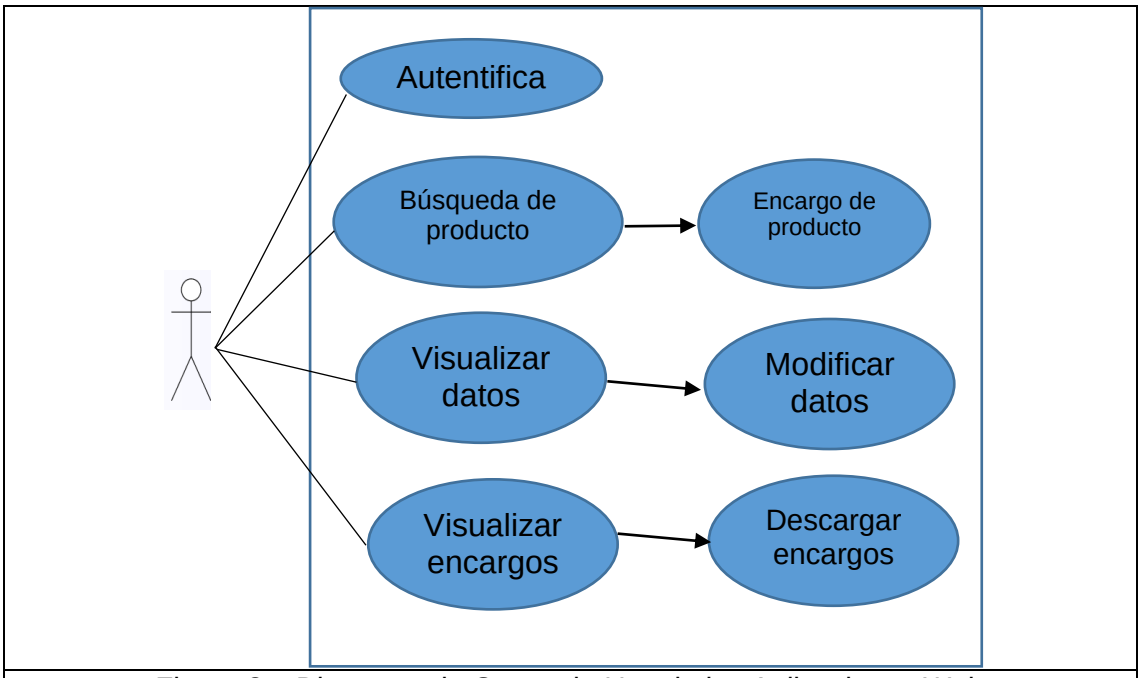


Figura 3a: Diagrama de Casos de Uso de las Aplicaciones Web

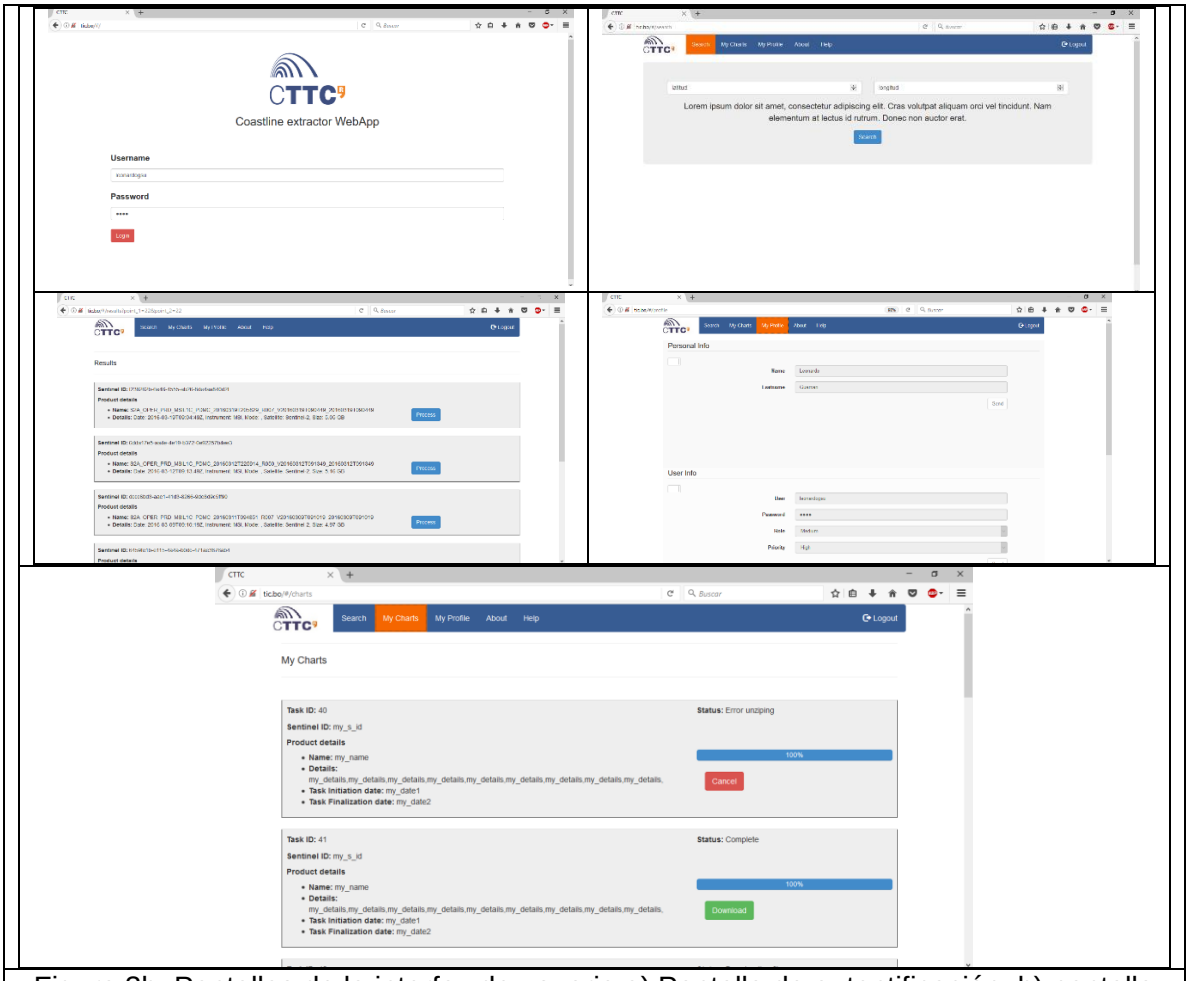


Figura 3b: Pantallas de la interfaz de usuario a) Pantalla de autentificación, b) pantalla de búsqueda, c) pantalla de resultados, d) pantalla My Profile, e) Pantalla My Charts en donde se visualiza el estado de las tareas

La interfaz de usuario se ejecuta sobre un navegador web (cliente) y genera peticiones para satisfacer los casos de uso sobre un una comunicación cliente-servidor utilizando transacciones HTTP (*Hypertext Transfer Protocol*).

HTTP provee un protocolo de trasferencia de información a nivel de aplicación (modelo OSI), define la sintaxis y la semántica que utilizan los elementos en una arquitectura web para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor.

Las respuestas a las peticiones que devuelve el servidor web contienen además del código devuelto por el servidor y de los encabezados, una estructura de datos del tipo JSON que contiene los datos solicitados por la interfaz de usuario o una respuesta de la solicitud de ejecutar alguna acción. En el Anexo 2 se detallan los tipos de respuestas JSON que se le envía al cliente web.

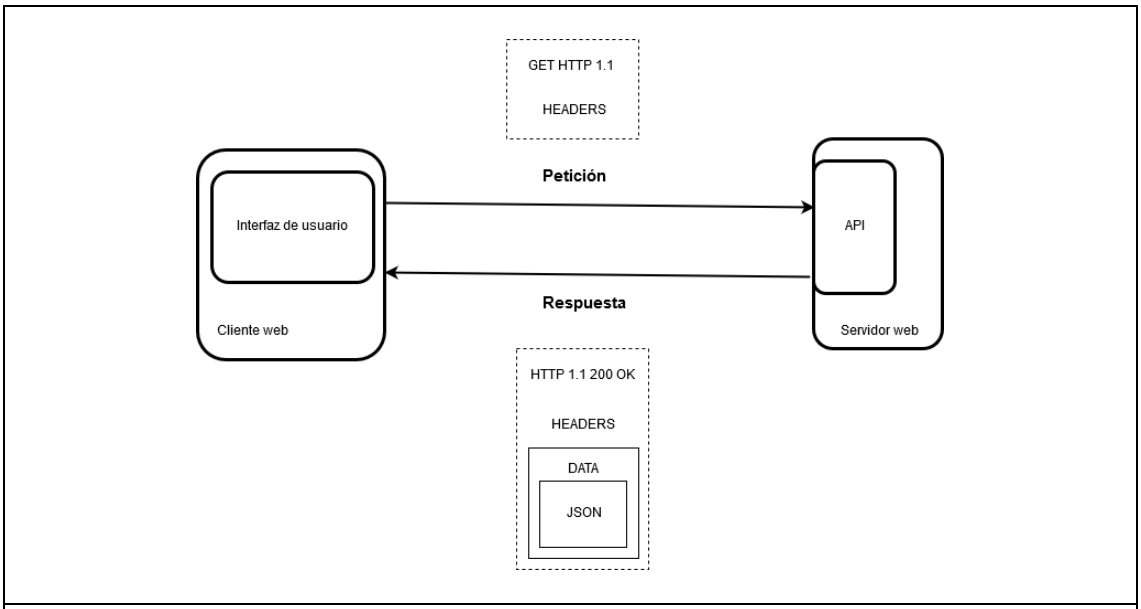


Figura 4: Peticiones y respuestas cliente servidor a través de la API del sistema

2.7.2.- Servidor web

El servidor web principalmente gestionará los procesos que intervienen en la comunicación entre la interfaz con sí mismo a través de REST (*Representational State Transfer*) [4], que agrupa un conjunto de protocolos y formatos de comunicación que definirán a la API del sistema.

REST se basa en el punto de acceso URI (*Uniform Resource Indentifier*), el cliente y servidor están físicamente separados y los enlaza únicamente un canal de comunicaciones HTTP [17]. El cliente no necesariamente necesita conocer cómo el servidor gestiona los datos, tampoco cómo los genera y el servidor no conoce de cómo el cliente presenta la información. Además el servidor no mantiene ningún estado de las peticiones del cliente.

Por lo mencionado anteriormente, el diseño del servidor web debe satisfacer las siguientes funciones:

- **Mapeo de los *endpoints* de entrada a la API del sistema**: Los *endpoints* definen la interfaz entre el cliente y los recursos del servidor. Están identificados de forma univoca mediante URIs. En la Tabla 2 se lista los *endpoints* utilizados.

- **Enviar consultas a *"Sentinel API Hub"*:** Las consultas que el servidor web realiza a la API de Sentinel *"Sentinel API Hub"* se lanzan cada que la interfaz de usuario lanza una petición de búsqueda, estas consultas se llevan a cabo a través URIs contra *"Sentinel API Hub"* que como respuesta, devuelve una lista de imágenes Sentinel-2 contenidas en un fichero XML.

- **Procesar las respuestas *"Sentinel API Hub ":*** el sistema deber procesar la respuesta XML que entrega *"Sentinel API Hub"* y trasformar esa respuesta en una estructura JSON

- **Generar respuestas a las peticiones de la interfaz de usuario.**

- **Generar tareas en el sistema.**

- **Enviar notificaciones al demonio.**

2.7.2.1.- Definición de endpoints

La API del sistema engloba un conjunto de funciones que permiten a la interfaz de usuario interactuar con el sistema a través de los endpoints, por tanto para este proyecto, un endpoint es la interface a través de la cual los sistemas externos (como la interfaz de usuario) pueden enviar y recibir mensajes. Los endpoints están implementados a través de URIs

En la tabla siguiente listamos los endpoints soportados actualmente por el sistema, en ella se pueden observar el método utilizado, endpoints, el uso, los parámetros enviados y el tipo de respuesta que retorna la API del sistema.

MÉTODO	ENDPOINT (PATH/RECURSO)	USO	RETORNA
POST	<HOST>:<PORT>/<API_PATH>/login	Autentificación	Objeto JSON
GET	<HOST>:<PORT>/<API_PATH>/products/search	Búsqueda de productos	Objeto JSON
GET	<HOST>:<PORT>/<API_PATH>/tasks/create	Generar línea de costa de prtucto determinado	Objeto JSON
GET	<HOST>:<PORT>/<API_PATH>/tasks/get	Listar tareas de usuario	Objeto JSON
GET	<HOST>:<PORT>/<API_PATH>/user/get	Listar atributos susuraio	Objeto JSON
GET	<HOST>:<PORT>/<API_PATH>/tasks/cancel	Cancelar tarea	Objeto JSON
GET	<HOST>:<PORT>/<API_PATH>/products/download	Descarga de producto	Enlace de descarga
GET	<HOST>:<PORT>/<API_PATH>/logout	Autentificación	Objeto JSON

Tabla 2: Lista de endpoints de la API

2.7.2.2.- Definición de consultas a la API de Sentinel

"Sentinel API Hub" provee una serie de URIs para realizar consultas y como respuesta a cada consulta devuelve un fichero XML del tipo *feed*.

La URI utilizada por el sistema posee cuatro partes:

- Service root URI
- Resource path
- Query
- Options

Es así que las peticiones de búsqueda que la API del sistema realizará al "*API Hub"* de Sentinel se realizan a través de un a consulta que se describe en la tabla siguiente:

SERVICE ROOT URI	https://scihub.copernicus.eu/dhus
RESOURCE PATH	/search
QUERY	?q=footprint:%22Intersects(<POINT_1>,<POINT_2>)%22AND%22platformname:%22Sentinel-2%22%22
OPTIONS	&$format=xml

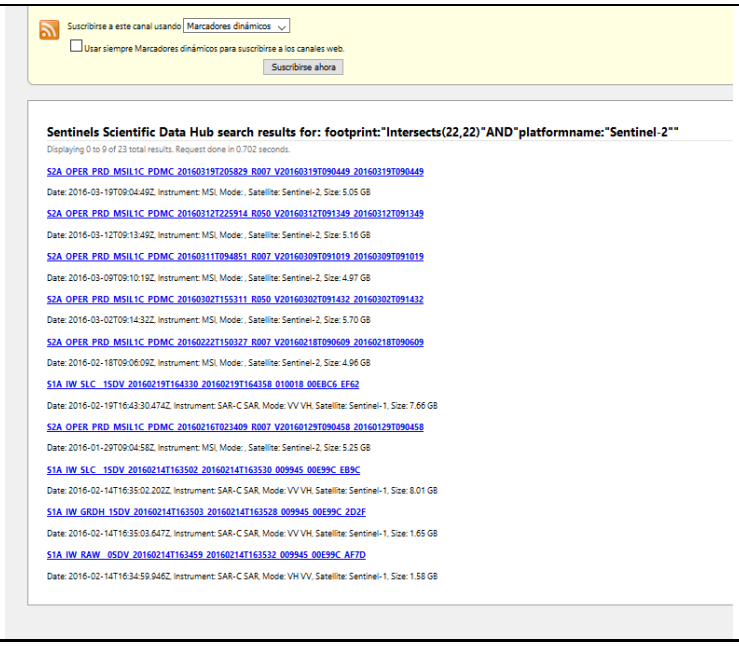
Tabla 3: URI de consulta del "*API Hub"* de Sentinel

En donde "<POINT_1>" y "<POINT_2>" son los valores de la longitud y latitud respectivamente, expresados en grados decimales, por ejemplo (41.3818, 2.1685).

2.7.2.3.- Definición de respuestas JSON

El servidor web también procesa las respuestas que entrega el *"Sentinel API Hub",* estas respuestas por defecto tienen el formato de redifusión *Atom* [6] que es un documento XML que describe colecciones de información relacionada denominadas "*feeds*".

En el siguiente cuadro (izquierda) se muestra un ejemplo de la respuesta que Sentinel API Hub entrega a la consulta de la tabla anterior. En la parte derecha observamos la estructura XML de resumida de la respuesta.




```
<xml>
  <entry>
    <title>TITLE</title>
    <id> ID</id>
    <summary>SUMMARY</summary>
    <str
name="platformname">PLATAFORM</str>
    <date
name="beginposition">DATE</date>
    <date
name="endposition">DATE</date>
    <str name="size">SIZE </str>
 </entry>
</xml>
```

Figura 5 : Ejemplo de respuesta ATOM que entre Sentinel API Hub

El servidor web debe generar una serie de respuestas en función a las peticiones que, en este caso, la interfaz de usuario realice. Las peticiones se han descrito en la Tabla 2 de la Sección 2.7.2.1, en donde por cada petición se lista el método, los endpoints, su función, los parámetros necesarios y el tipo de respuesta que la API del sistema genera.

Existen en general cinco tipos de respuestas que la API del sistema entrega, para más detalles favor ver el Anexo 3.

TIPO DE RESPUESTA	DESCRIPCIÓN
Respuesta JSON 0	Respuesta a una petición que no se puede realizar.
Respuesta JSON 1	Respuesta a una petición realizada exitosamente
Respuesta JSON 2	Respuesta a una autentificación exitosa.
Respuesta JSON 3	Respuesta a una búsqueda exitosa.
Respuesta JSON 4	Respuesta a una solicitud de listar tareas exitosas.
Respuesta JSON 5	Respuesta a una solicitud de listar información de usuario exitosa.

Tabla 4: descripción del tipo de respuesta que la API del sistema genera

2.7.2.4.- Generar Tareas

Definimos como tarea, a un "encargo" para generar la línea de costa de una imagen Sentinel-2 mediante la ejecución de una serie de procesos. Dichos procesos permitirán la tarea pase de un estado a otro hasta generar la línea de costa.

Cada tarea tiene los estados descritos en la Tabla 5 y Tabla 6 en donde "x" representa uno de los 3 estados intermedios (x={01,02,03}) .

ESTADO	DESCRIPCIÓN
1xx	Descarga
2xx	Descompresión
3xx	Extracción
4xx	Cambio formato
5xx	Extracción línea de costa
6xx	Compresión
7xx	Notificación

Tabla 5: Estados generales de la tarea

ESTADOS INTERMEDIOS	DESCRIPCIÓN
00	En cola
01	En progreso
03	Con error

Tabla 6: Estados intermedios de la tarea

En la sección 2.7.3 se realizará una explicación más completa sobre los estados y su relación con cada etapa del proceso de generación de línea de costa.

2.7.2.5.-Enviar notificación al demonio

Cada vez que se crea una tarea nueva, el servidor web lo notifica al demonio a través de una notificación TCP que le indica que es necesario ponerse en marcha.

La notificación se envía a través de una conexión TCP cliente servidor, mediante un puerto determinado, (el sistema por defecto utiliza el puerto TCP 2000) en donde la el servidor web hace de cliente y el demonio es el servidor TCP.

Este tipo de notificación no espera respuesta por parte del demonio y su único objetivo es indicarle que hay tareas pendientes por llevar a cabo.

2.7.3.- Demonio TCP

Una vez el servidor web haya enviado la notificación TCP, el demonio debe gestionar y lanzar una serie de procesos en un orden específico hasta generar la línea de costa de la imagen Sentinel encargada por el usuario.

Los procesos diseñados e implementados además, envían notificaciones al demonio cuando sucede un evento. El evento se produce cuando se genera una tarea nueva o un proceso ha llegado a su fin, debido a un error o su finalización exitosa.

Así pues, el servidor web y dos procesos deben notificar al demonio cada vez que ha ocurrido algo, por tanto debido a la naturaleza asíncrona de estos eventos, el demonio debe contar con servidor TCP multihilo capaz de atender varias conexiones a la vez.

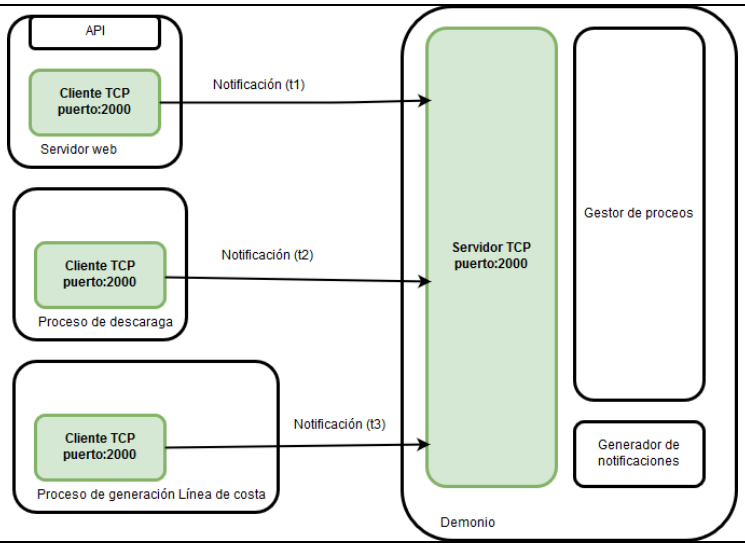


Figura 6: El demonio cuenta con un servidor TCP "multiconexión" capaz de atender varias conexiones a la vez

El servidor TCP implementado en el demonio atiende las notificaciones enviadas de:

- El servidor web cuando se genera una nueva tarea es decir cuando se ha encargado la generación de una línea de costa de una porción de terreno.

- El proceso de descarga cuando ha habido un error o cuando ha terminado de descargar las imágenes Sentinel-2.

- El generador de línea de costa, cuando suceda un error o cuando la extracción de la línea de costa se haya ejecutado y finalizado correctamente.

Una vez el demonio ha recibido una notificación, este lee la base de datos y confecciona una lista de tareas a ejecutar.

Cada tarea corresponde a un encargo de generación de línea de costa. Cada tarea posee además, un estado que indica al demonio (y por tanto sistema en general), qué pasos seguir qué procesos lanzar para atender dicha tarea (Figura 7).

Una vez el demonio ha confeccionado una lista con N tareas, empieza a ejecutar la primera tarea en la lista, el orden está dado por el estado y la prioridad de la tarea. Los estados del tipo x01 prevalecen sobre los del tipo x03 (ver Tabla 6 y Tabla 6 para la definición de los códigos.).

Cuando el demonio empieza a tratar la primera tarea de la lista (Figura 8) verifica el estado de la misma, si la tarea no ha sido atendida su estado será de "100" y las tareas que se encuentren en estado "100" son atendidas por el proceso número "1", el proceso de descarga. Entonces el demonio lanza los procesos de descarga, si el proceso se ejecuta de forma exitosa, el demonio

actualiza el estado de la tarea en la base de datos con un valor de “101”, que indica que el proceso de descarga se está ejecutando.

Por el contrario, si el proceso no ha sido ejecutado correctamente, el demonio actualiza el estado de la tarea con un valor de “103”.

El proceso de descarga lanzado es independiente al demonio, este se encargará de notificarle cuando la descarga se haya finalizado y realizar la debida actualización de estados en la base de datos.

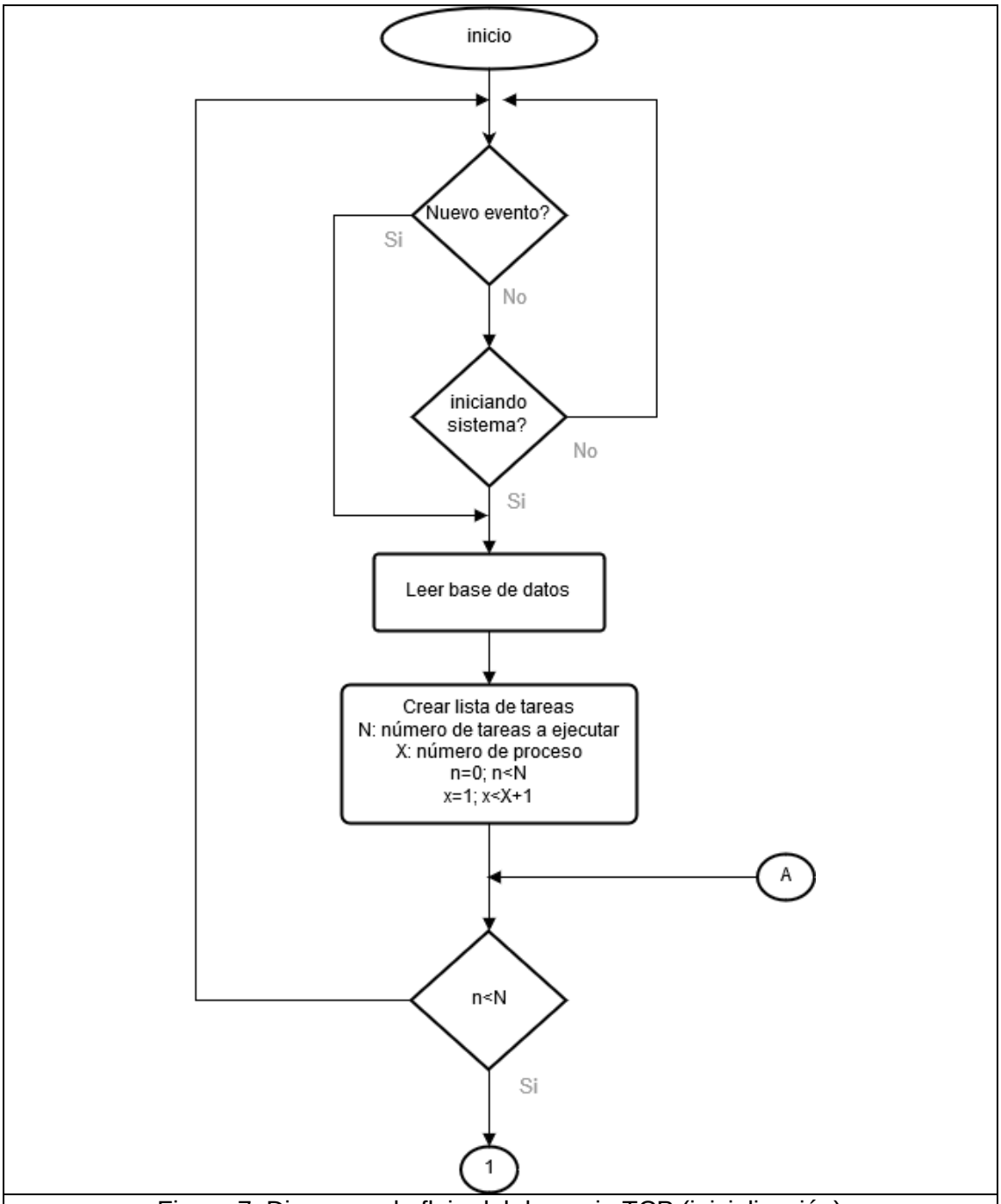


Figura 7: Diagrama de flujo del demonio TCP (inicialización)

La primera tarea ya ha sido atendida, por tanto si no hay más tareas el sistema se quedará esperando que exista un nuevo evento, por ejemplo una nueva tarea creada por el servidor web, la finalización del proceso de descarga o la finalización del proceso de generación de línea de costa.

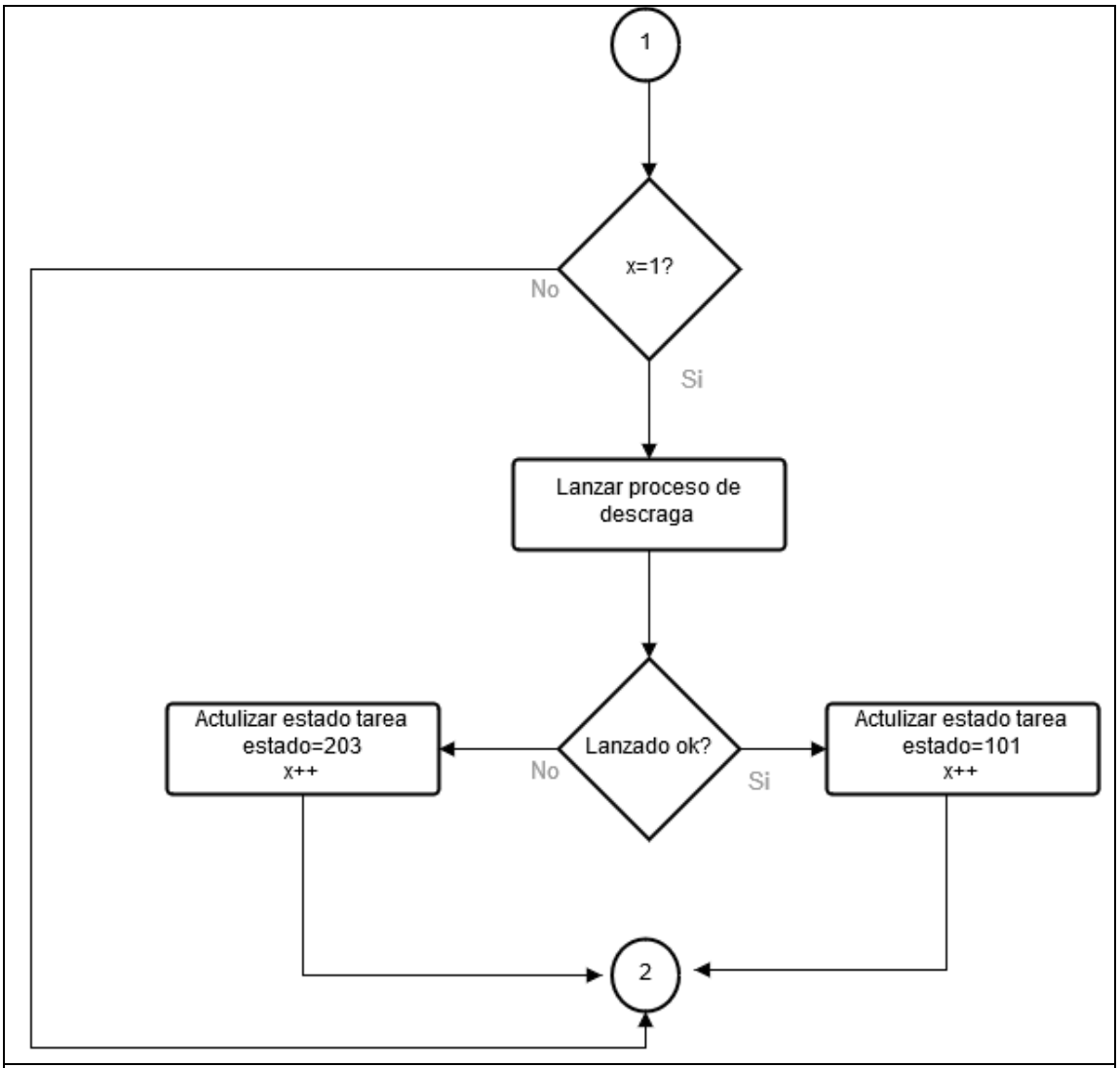


Figura 8: Diagrama de flujo del demonio TCP (gestión proceso de descarga)

Si existiera una nueva tarea que se ha creado antes de que el demonio termine la descarga de la anterior tarea, el demonio lanzaría otro procesos de descara. El número de procesos de descarga que se pueden lanzar de forma paralela es configurable, por defecto, sólo se pueden lanzar 2 procesos de descarga.

Si llega una tarea de la lista con un estado igual a “200” (Figura 9), el proceso que el demonio lanza es el proceso número “2”, correspondiente al proceso de descompresión. El demonio lanza el proceso, si no hay error actualiza el estado de la tarea a “201”, si lo hay “203” y se pasa a atender a la siguiente tarea.

Si el proceso de descarga se lanza correctamente, el demonio espera su finalización o un *time out*. Si salta el *time out* se ha producido un error y se pasa a atender la siguiente tarea. Si el proceso de descompresión finaliza

correctamente, el demonio actualiza el estado de la tarea con un valor de “400”, luego se pasa a atender a la siguiente tarea de la lista.

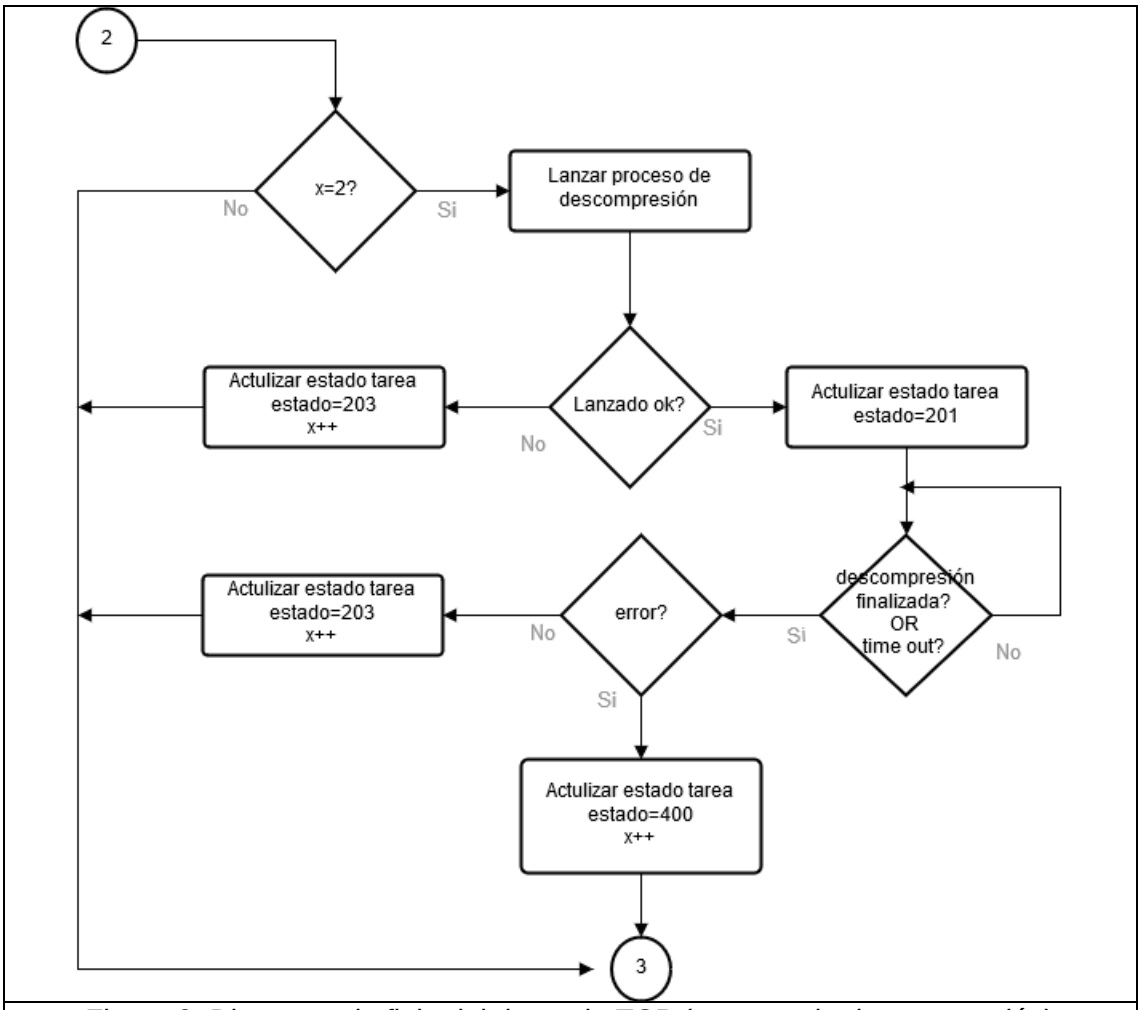


Figura 9: Diagrama de flujo del demonio TCP (proceso de descompresión)

Existe un proceso que no ha sido implementado (Figura 10), cuya funcionalidad es filtrar las imágenes descomprimidas en función de sus características para realizar otros procesos cartográficos. De momento el proceso “3” no realiza ninguna acción.

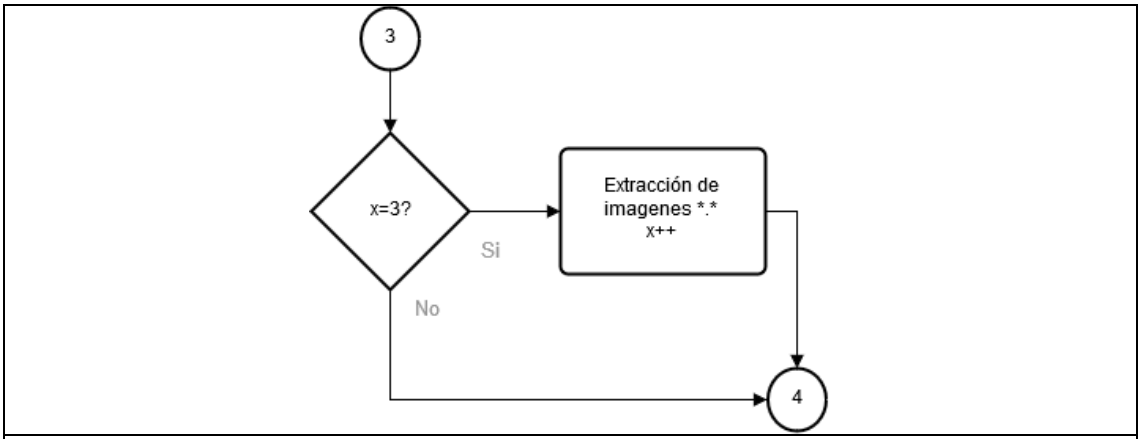


Figura 10: Diagrama de flujo del demonio TCP (proceso de descompresión)

Cuando una tarea tiene estado “400” el demonio lanza los procesos de conversión de formato JP2000 a JPG (Figura 11). El demonio en primer lugar crea una lista de 13xM imágenes a convertir, en donde M es el número de porciones de terreno de 500m^2 además cada porción de terreno posee 13 imágenes que corresponden a exploraciones de distintas bandas (ver Anexo 1).

Es así que el demonio debe lanzar 13xM veces el proceso de conversión de imágenes por tarea (o encargo) y controlar el estado y progreso de cada proceso.

Es importante señalar que mientras el demonio está ocupado ejecutando alguna tarea o esperando la finalización de otra tarea, los procesos lanzados funcionan de forma independiente y paralela. El límite está dado por la capacidad de procesamiento del hardware sobre el cual el sistema se ejecute.

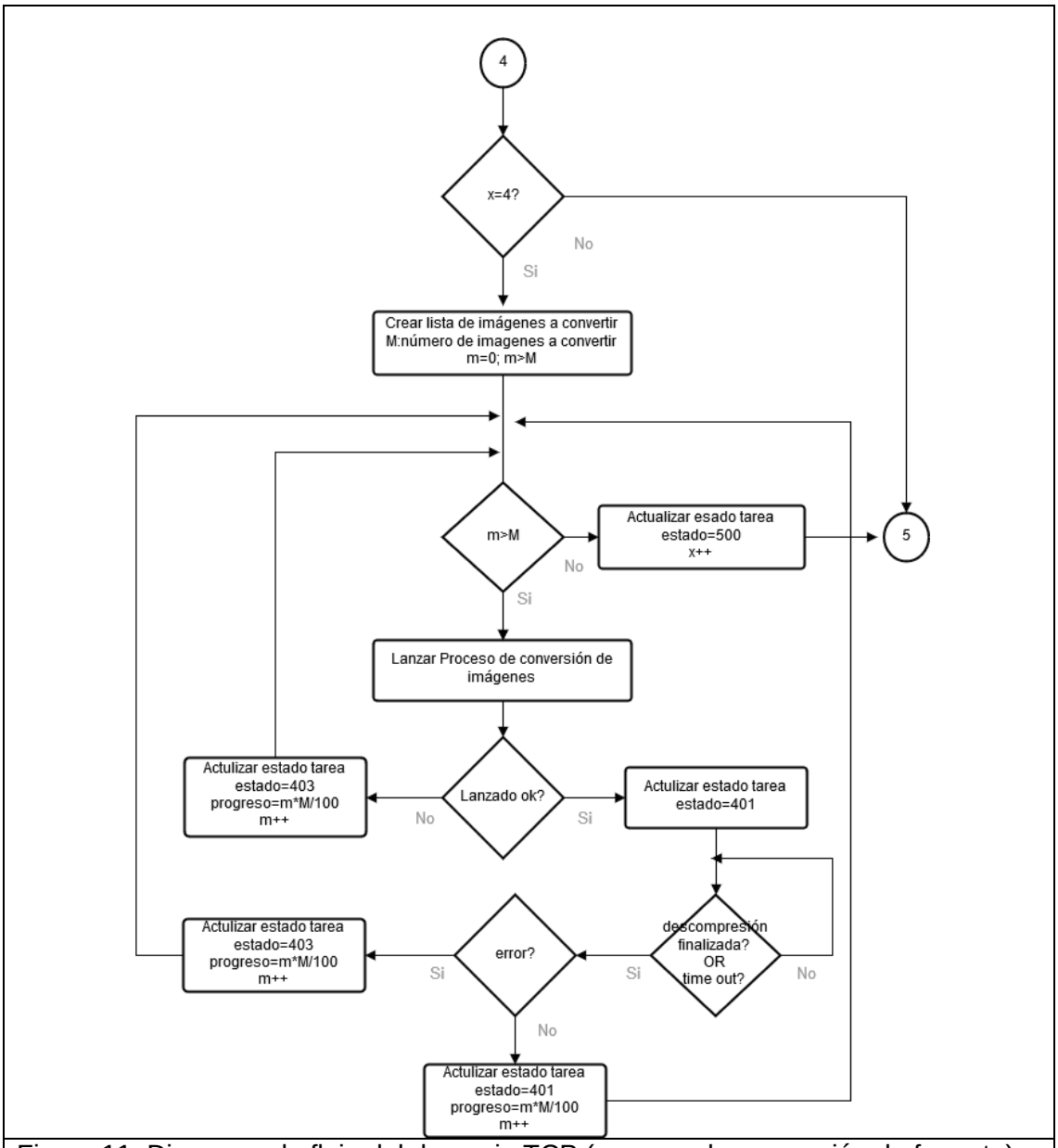


Figura 11: Diagrama de flujo del demonio TCP (proceso de conversión de formato)

Si el estado de la tarea es “500” el proceso que el demonio lanza es el proceso de generación de línea de costa. De forma similar al procesos de conversión de formato, el demonio debe lanzar M veces el proceso y controlar si lo ha lanzado de forma correcta. El proceso completo se observa en la Figura 12.

Cuando una tarea tiene el estado “600” (Figura 13) el demonio lanza el proceso de compresión realizada por el programa 7-Zip, y su control se realiza de la misma forma que en el caso de descompresión, pero actualiza los valores de estado con un “700” cuando se realiza la compresión correctamente y “603” si no lo hace.

El compresor recibe M imágenes para comprimir y entregará una carpeta en formato *zip*.

Cuando la tarea tenga estado 700, significa que el encargo de generación de línea de costa se encuentra finalizado, y el demonio genera un email para notificarlo al usuario.

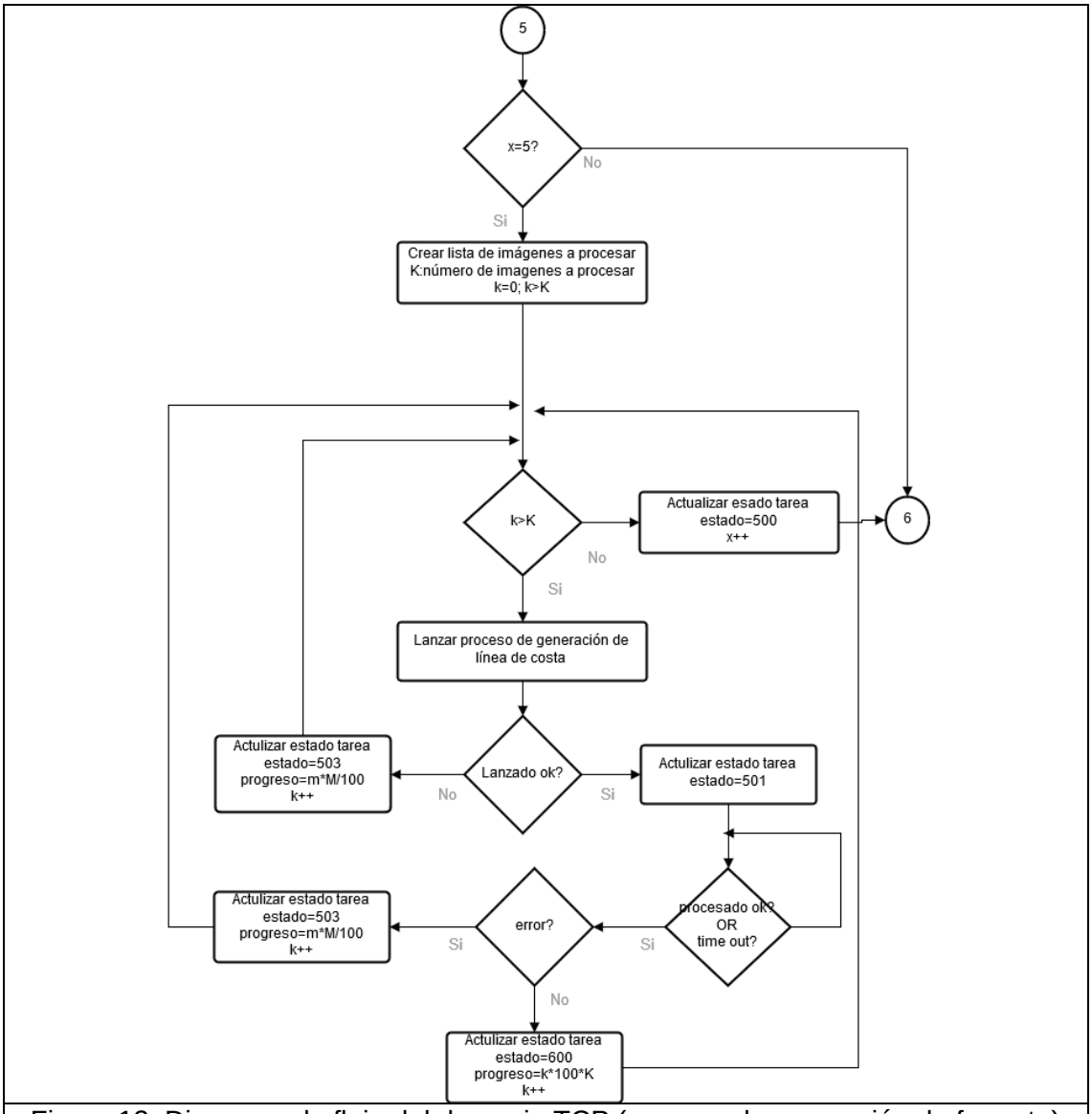


Figura 12: Diagrama de flujo del demonio TCP (proceso de conversión de formato)

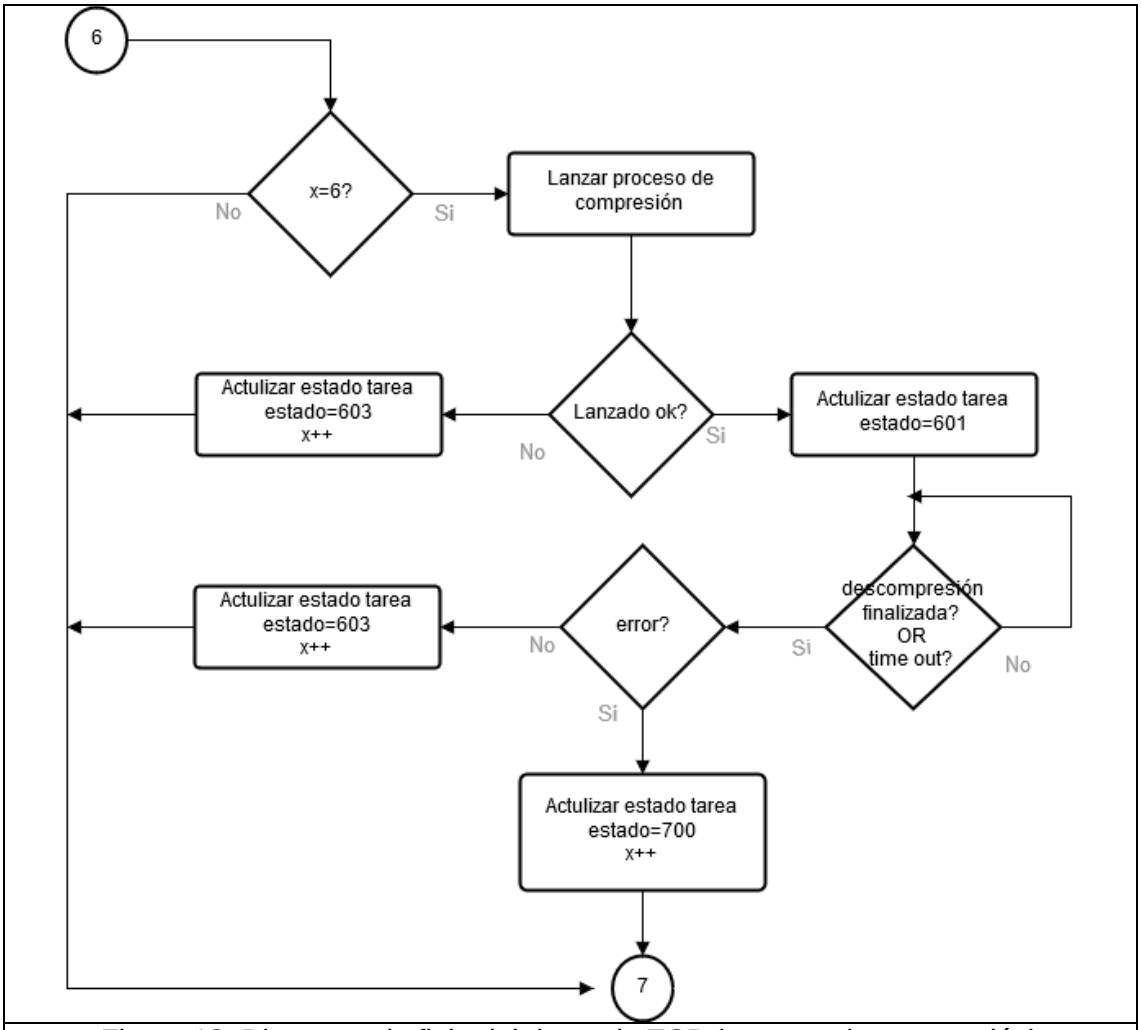


Figura 13: Diagrama de flujo del demonio TCP (proceso de compresión)

Finalmente si no hay más tareas en la lista, el demonio espera a que exista un nuevo evento a través de una notificación TCP para repetir todo el proceso descrito (Figura 14).

Si existiese más tareas en la lista a ser realizadas el contador “n” se incrementaría en una unidad y el proceso se repetiría para atender la siguiente tarea de la lista.

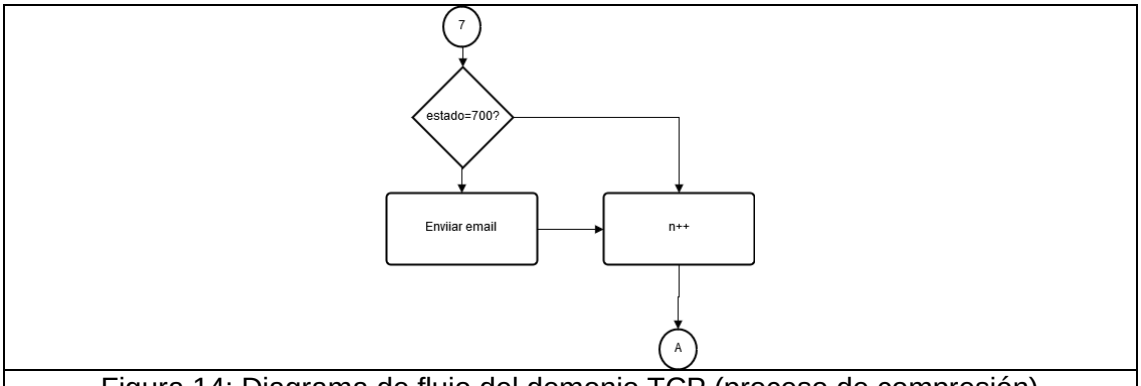


Figura 14: Diagrama de flujo del demonio TCP (proceso de compresión)

2.7.4.- Procesos

Los procesos corresponden a la parte del sistema que realiza el trabajo que, en un determinado orden de ejecución, permiten obtener el producto final (un plano con línea de costa).

Como se dijo en la Sección 2.7.3, cuando la interfaz de usuario envía una petición para crear una nueva tarea al servidor web (a través de la API del sistema), el servidor web crea una nueva tarea añadiendo un registro en la base de datos y notifica al demonio. Luego el demonio lee la base de datos y elabora una lista de tareas a ejecutar. La lista depende de la disponibilidad de recursos del sistema y esta disponibilidad es configurable a través de un fichero de configuración general del sistema.

Una vez el demonio haya elaborado la lista de tareas, este lanzará procesos en función del estado en que se encuentre cada tarea y su prioridad. El siguiente cuadro describe la relación de estados de las tareas con los procesos que el demonio lanza.

ESTADO	PROCESO A LANZAR	Nº PROCESO
100 o 103	Proceso de descarga	1
200 o 203	Proceso de descompresión	2
300 o 303	Proceso de extracción	3
400 o 403	Proceso de conversión de formato	4
500 o 503	Proceso de generación línea de costa	5
600 o 603	Proceso de compresión	6
700 O 703	Proceso de notificación	7

Tabla 7: Procesos que lanza el demonio según el estado de la tarea

Todos los procesos siguen un flujo más o menos común: al ser lanzados, verifican que los parámetros que han recibido sean los correctos, si no lo son, finalizan su ejecución con un valor de *exit code* mayor a cero.

Si los parámetros son correctos, los procesos inician su trabajo hasta finalizarlo. Si lo finalizan debido a un error, terminan con un valor de *exit code* mayor a cero, si no con un valor de *exit code* igual a cero.

El *exit code* es un código que un proceso en los sistemas operativos de la familia UNIX, retorna cuando se ha ejecutado, si se ejecuta de forma exitosa retorna “0”, caso contrario retorna un valor mayor a cero, que usualmente se interpreta como un código de error. Gracias a este código de salida el demonio se informará si el proceso se ha lanzado correctamente o no.

Los procesos de descarga y de generación de línea de costa, (a diferencia del procesos de compresión/descompresión y conversión de formato), actualizan periódicamente la base de datos informando sobre su progreso, además cuando su ejecución finaliza, también actualizan la base de datos con los respectivos códigos de estado.

PROCESO	ARGUMENTOS ESPERADOS	DATOS DE ENTRADA	DATOS DE SALIDA
Descarga	1 Id de tarea	URI de descarga del fichero	Fichero zip de unidades de GB de tamaño
Descompresión	1 Carácter “e” 2 Ruta fichero a descomprimir 3 Cadena “*.jp2” 4 Cadena “-r” 5 “-o”+ Ruta de destino de descompresión	Fichero zip de unidades de GB de tamaño	Conjunto de imágenes JP2000
Conversión de formato	1 Ruta de imagen JP2000 2 Ruta de imagen JPH	Imagen JP2000	Imagen JPG
Generación LC	1 Ruta imagen 2 Ruta destino	Imagen JPG	Imagen JPG con línea de costa generada
Compresión	1 Carácter “a” 2 Cadena “-r” 3 Ruta destino 4 Ruta origen	Conjunto de imágenes JPG	Fichero ZIP

Tabla 8: Argumentos, datos de entrada y datos de salida de cada proceso

2.7.4.1.- Proceso de descarga

La descarga de las imágenes es la primera etapa de la tarea de generación de línea de costa, para llevarla a cabo se ha diseñado he implementado un proceso de descarga diseñado específicamente para cumplir las necesidades del sistema.

El proceso de descarga se lanza cuando el estado de la tarea sea “100”. Para lanzarlo el demonio debe enviar junto al comando de ejecución un argumento en este caso es el ID de la tarea (ver Tabla 8).

Si los argumentos recibidos por el proceso de descarga no son los correctos, el procesos termina su ejecución con un *exit code* mayor a cero, entonces el demonio debe ser capaz de capturar este valor y actualizar el estado de la tarea a “103”

Por el contrario, si los argumentos son los correctos el proceso de descarga se conecta a la base de datos y actualiza el estado de la tarea a “101”.

El proceso de descarga podría llegar a tardar algunas horas en descargar todo el fichero desde los servidores de Sentinel, esto depende de la velocidad de

conexión entre el sistema y los servidores de Sentinel y de su disponibilidad. Por este motivo, este proceso actualiza periódicamente el progreso de la descarga.

Si la descarga llega a su fin correctamente, el proceso actualiza la base de datos con un estado de la tarea igual a “200”, y envía una notificación TCP al demonio y finaliza su ejecución.

En caso de que la descarga haya finalizado debido a un error, el proceso actualiza en la base de datos el estado de la tarea con un valor igual a “103” y finaliza su ejecución.

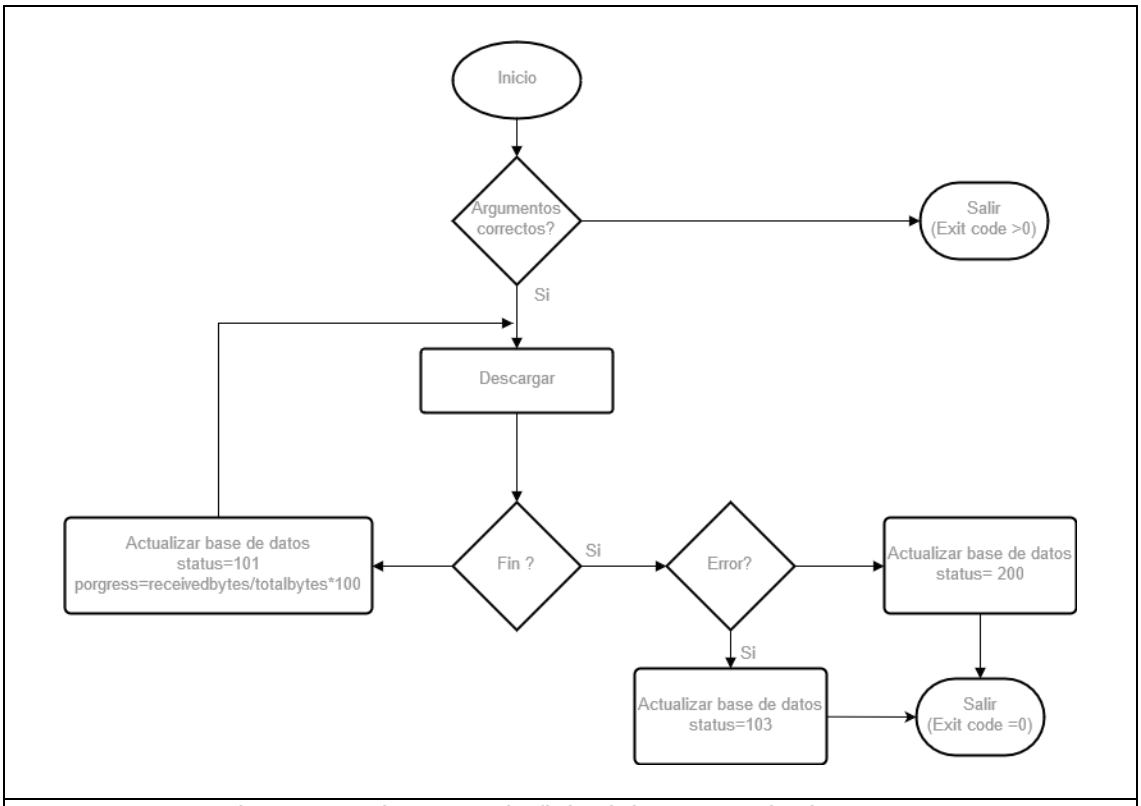


Figura 15: Diagrama de flujo del proceso de descarga

2.7.4.1.- Proceso de descompresión

Para descomprimir las imágenes de hace uso del programa 7-Zip. Se lo lanza con los argumentos que se listan en la Tabla 8.

El primer argumento “e” representa el comando de extracción, la cadena “*.jp2” implementa un filtro apara que sólo sean extraídas las imágenes JP2000, “-r” es una opción que si está presente realiza una descompresión recursiva y finalmente “-o” indica al descompresor que le vamos a especificar una ruta de destino para los ficheros descomprimidos.

Para controlar si la ejecución de la descompresión se ha realizado correctamente el demonio se queda a la espera de que el proceso 7-Zip finalice

la descompresión. Si el *exit code* es cero, el demonio actualizará el estado de la tarea a “400”, si el *exit code* es mayor que cero, el demonio actualizará el estado con un “203”.

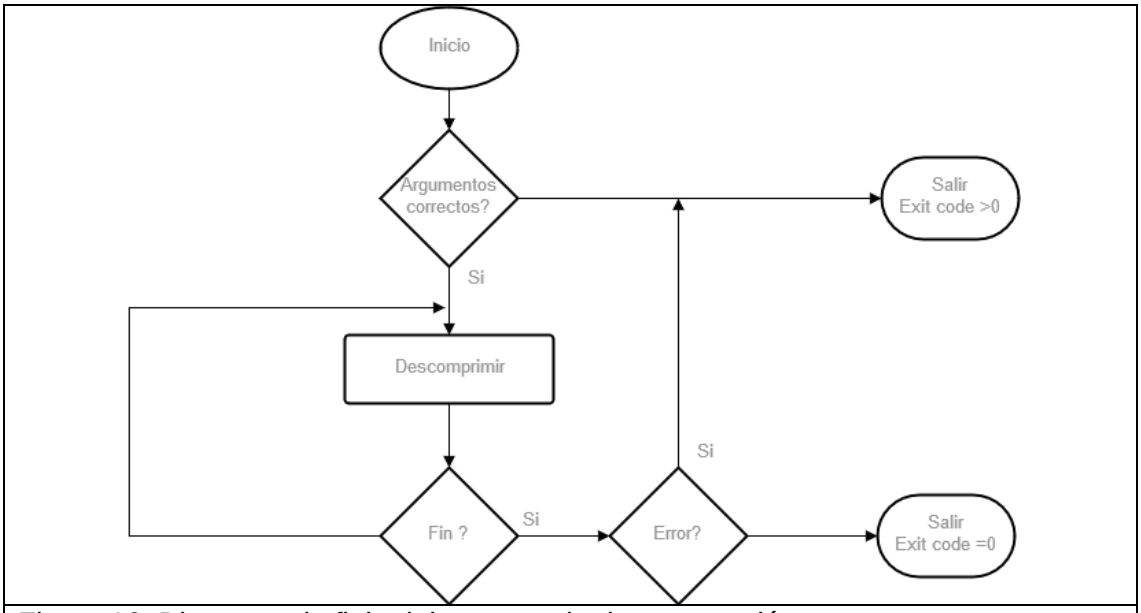


Figura 16: Diagrama de flujo del proceso de descompresión

2.7.4.2.- Conversión de formato

Las imágenes que proporciona “Sentinel API Hub” tienen el formato JP2 (JPEG 2000) y el proceso de generación de línea de costa trabaja con imágenes del tipo JPG, por tanto es necesario realizar el cambio de formato.

La etapa de descompresión (si no ha generado error) genera 13xN imágenes JP2000, estas imágenes corresponden a “N” porciones de terreno de 500 m^2 donde cada una de estas porciones de terreno posee 13 versiones correspondientes a las bandas espectrales (ver Anexo 1).

De este modo, el demonio lanzara 13xN veces sucesivas el programa que ejecuta la conversión de formato, cada proceso de conversión de imágenes se lanzará y controlará de la misma forma que lo hace en el proceso de descompresión.

De forma similar al proceso de descompresión, si es lanzado correctamente, el demonio actualiza el estado de la tarea con un valor “401” si no un “403”.Cuando las 13xN imágenes JP2000 se han convertido correctamente a JPG, el demonio actualizará el estado de la tarea con un valor de “500”.

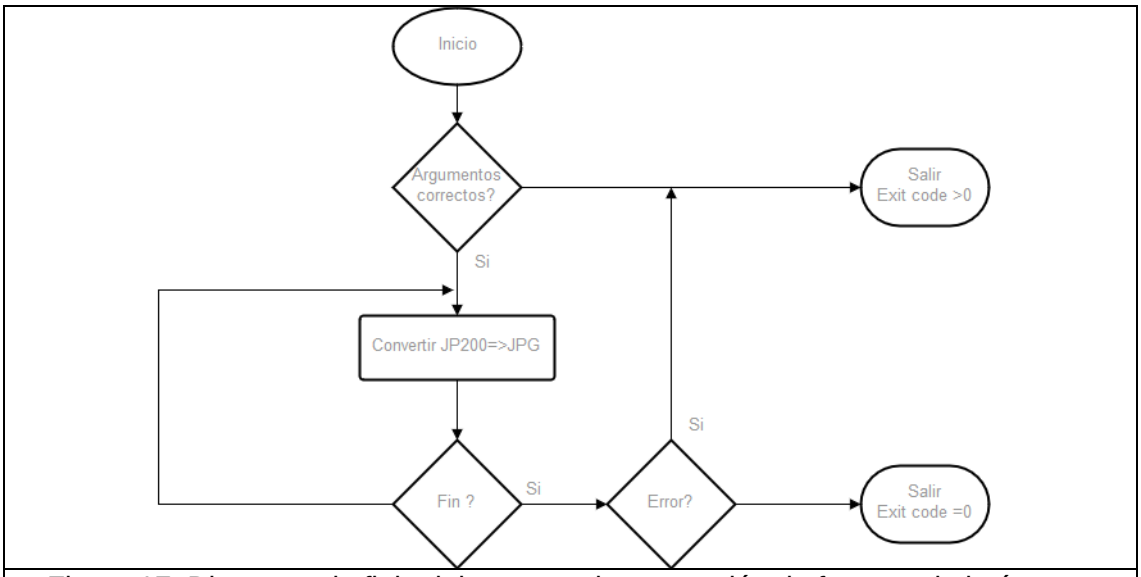


Figura 17: Diagrama de flujo del proceso de conversión de formato de imágenes JP2000 a JPG

2.7.4.3.- Generación de línea de costa

Las generación de la línea de costa se la realiza con un proceso implementado a partir de una clase diseñada por el CTTC, que a partir de tres tipos de imágenes Sentinel-2 correspondientes a las bandas B03, B08 y B11 (ver Anexo 1) generan una línea de costa que se imprime sobre una imagen de tipo B08 generando como salida una imagen cartográfica que contiene una línea de costa sobrepuesta.

El total de imágenes JPG que se procesan serán 3xN, el demonio es el responsable de lanzar N veces el proceso de conversión utilizando el programa ImageMagick, por consiguiente el resultado de esta etapa generará N imágenes.

Además el proceso recibe como parámetros de entrada la ruta de las imágenes del tipo B03, B08 y B11 en formato JPG. Si el proceso se ejecuta correctamente, cambia el estado de la tarea a “501”, si hay un error “503” y cuando finaliza la generación de línea de costa de las N imágenes, actualiza el estado de la tarea a “600”.

2.7.4.3.- Compresión de las imágenes

El proceso de compresión se lanza cuando el estado de la tarea sea igual a “600” a través de 7-ZIP. Al ser un programa que no tiene acceso a la base de datos del sistema, el demonio actualiza sus estados. Si lo lanza de forma correcta el estado de la tarea se actualiza con un “601”, si no con “603”. Como en el caso de la descompresión y de la conversión de formato, el demonio lee el *exit code* del proceso para saber si el proceso se ha lanzado correctamente o no.

El proceso de compresión recibe N imágenes correspondientes a las N porciones de terreno de 500m^2 y después de comprimirlas, entrega un fichero 7-Zip que contiene el producto final de la tarea.

Este fichero comprimido será entregado al usuario a través de un enlace de descarga.

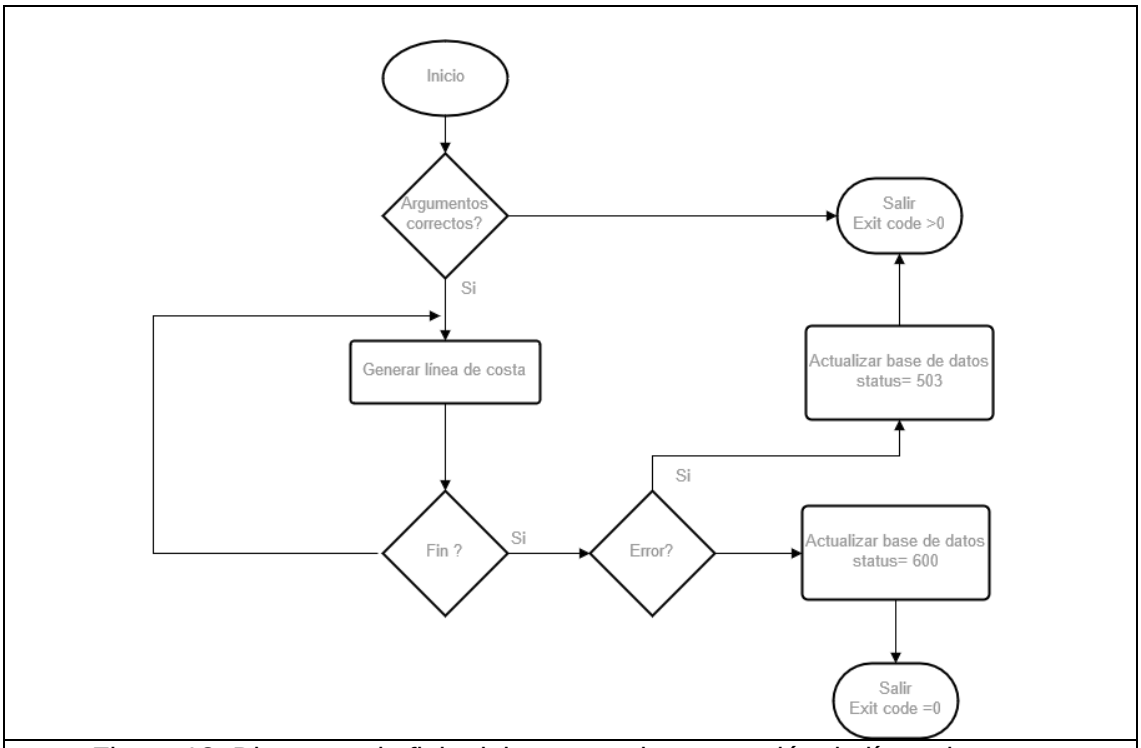


Figura 18: Diagrama de flujo del proceso de generación de línea de costa

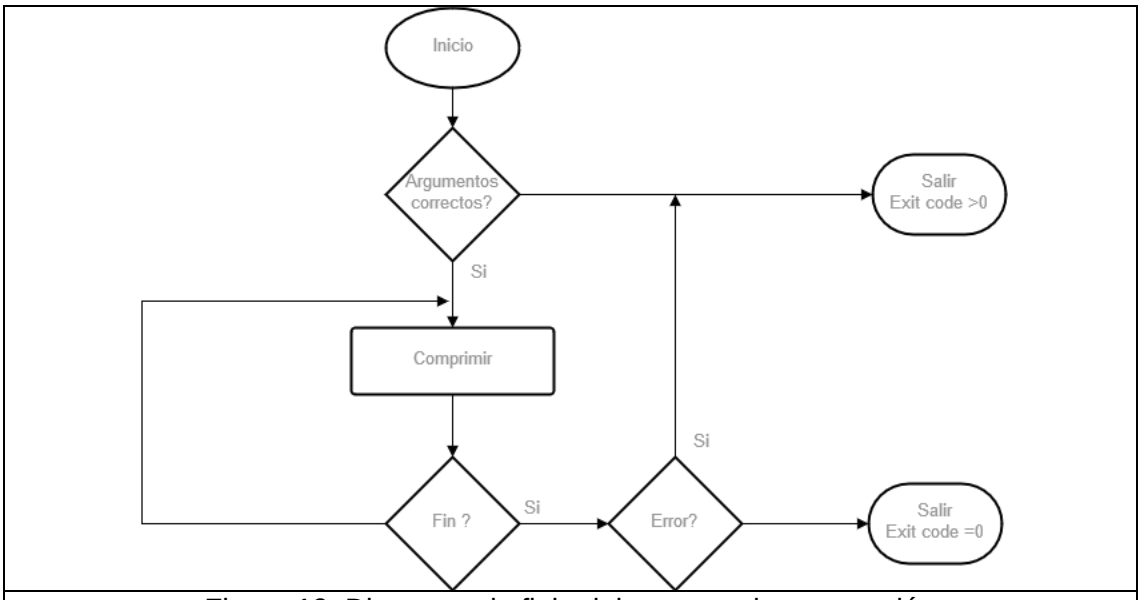


Figura 19: Diagrama de flujo del proceso de compresión

2.7.5.- Modelo de datos

La cohesión e integración del sistema que es objeto de diseño se consigue a través de la base de datos, en donde lo más importante es el modelo de datos.

Un modelo de datos es un conjunto de conceptos utilizados para organizar los datos de interés y describir su estructura en forma comprensible para un sistema.

El sistema utiliza cinco clases que describen los objetos con los que trabaja: Producto, Tarea, Usuario, Persona y un objeto que describe la información de contacto. Las relaciones y atributos de cada objeto se pueden observar en la figura siguiente.

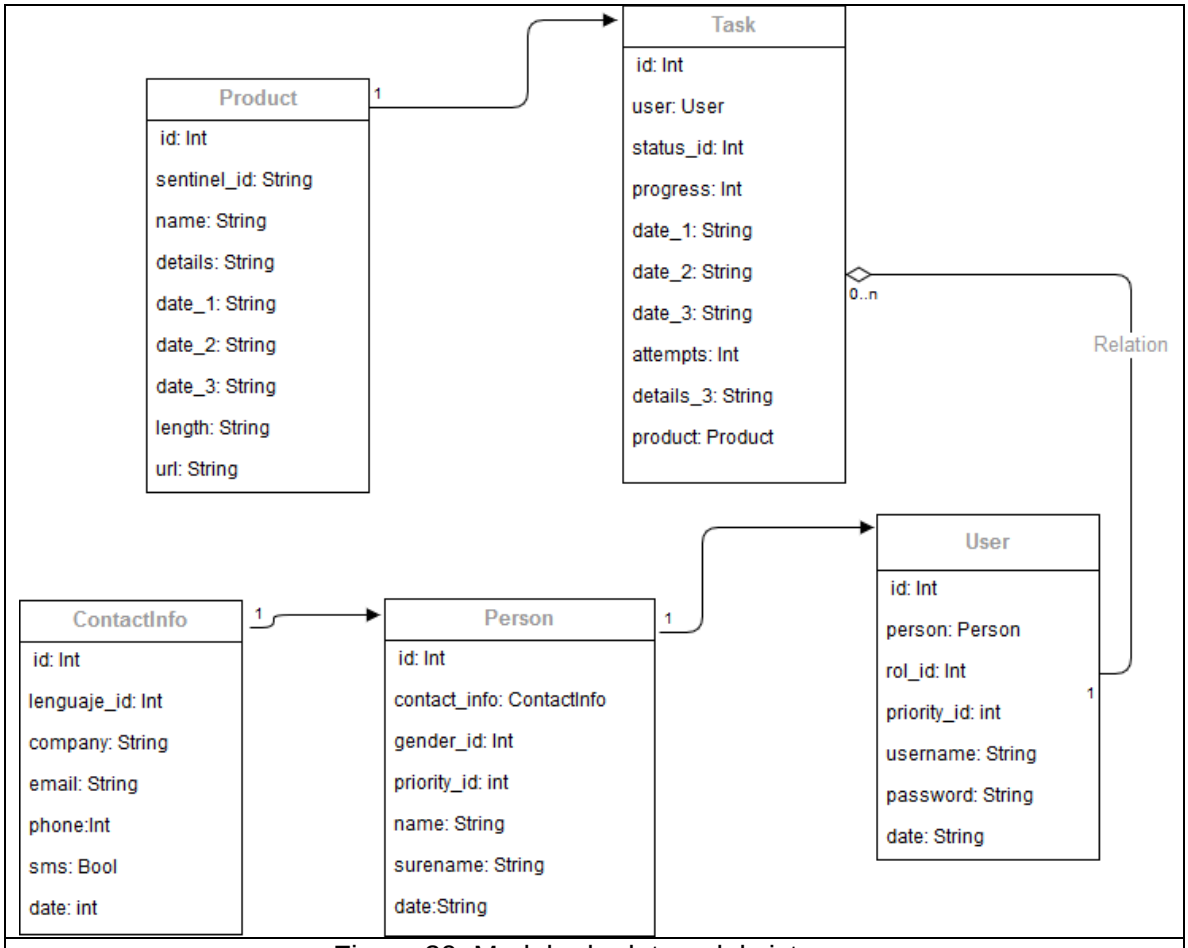


Figura 20: Modelo de datos del sistema

El modelo se implementará utilizando una base de datos relacional del tipo SQL (*Structured Query Language*), específicamente se utilizará MySQL. La relación de las tablas es la que se señala en la Figura 21.

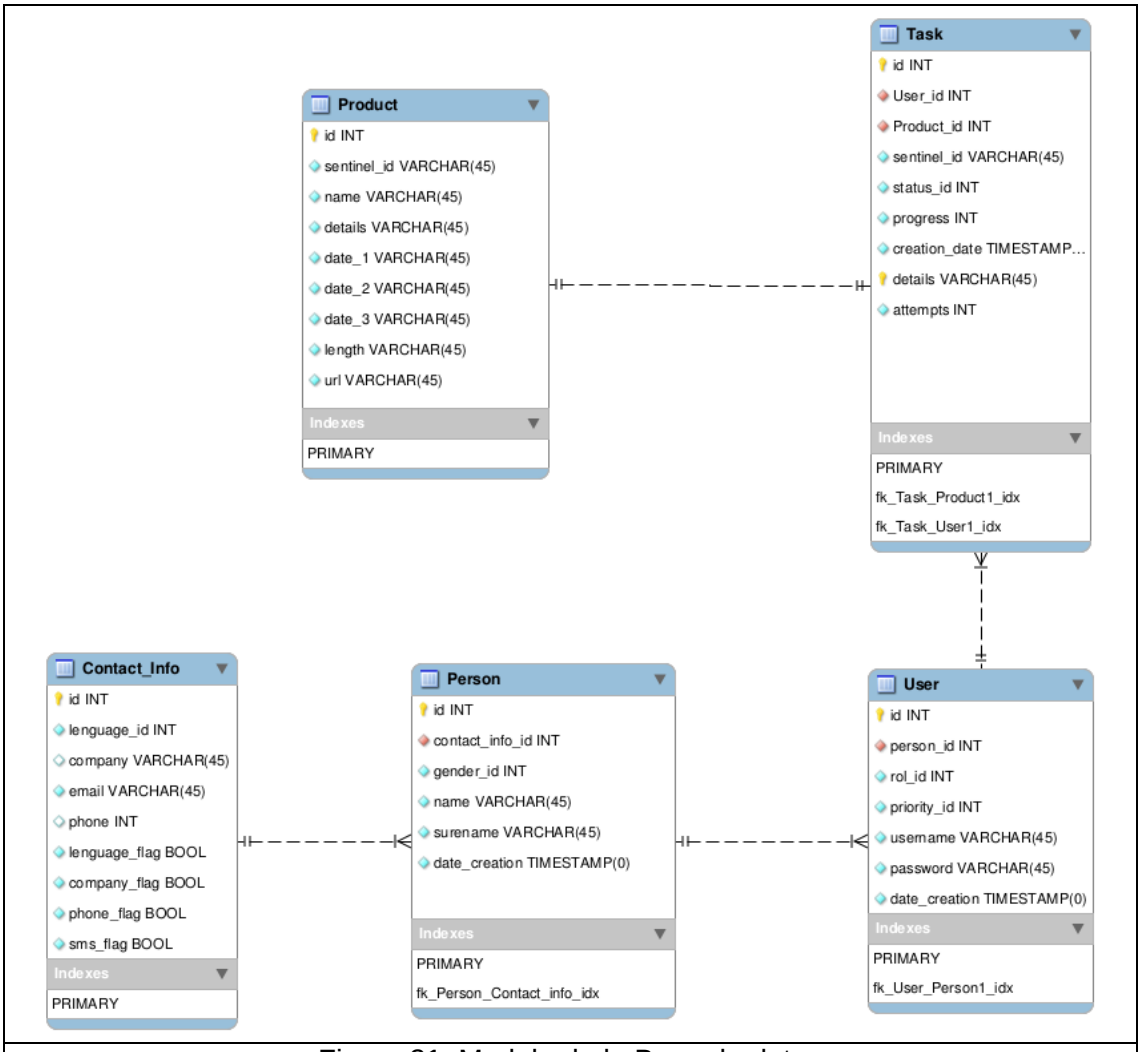


Figura 21: Modelo de la Base de datos

La base de datos es del tipo relacional, sin embargo el modelo del sistema utiliza objetos, por lo que es necesario realizar el diseño de una clase que mapee los objetos del sistema en tablas relacionales.

CAPÍTULO 3. IMPLEMENTACIÓN

3.1.- Introducción

En este capítulo se explicará cómo se ha implementado el diseño descrito en el Capítulo 2. En primer lugar describiremos el entorno de desarrollo que se ha utilizado, listando los programas y librerías necesarias para su puesta en marcha.

Posteriormente se describirán brevemente las aplicaciones utilizadas y finalmente se explicará cómo se han implementado cada uno de los elementos del sistema.

3.2.- Entorno de desarrollo

El prototipo diseñado e implementado ha sido desarrollado en un entorno Linux, específicamente se ha utilizado como sistema operativo Ubuntu 16.04. Para la mayor parte de la implementación se utilizado Qt 5.5.1 a través de su propio IDE denominado *Qt Creator*.

Por otro lado, han sido necesaria la instalación y compilación de librerías y programas para cubrir los requerimientos de las aplicaciones. En las tablas siguientes se señalan las librerías y aplicaciones utilizadas en el desarrollo del sistema.

NOMBRE	FUNCIÓN	¿COMPILACIÓN?	VERSIÓN
MySql Server	Base de datos	Sí	5.7.17
ImageMagick	Conversión de formatos	Sí	6.9.3-8
OpenCV	Procesado de la imagen	Sí	2.4.12
Rabbit VCS	Cliente SVN	No	
Doxigen	Generar documentación	No	
7-Zip	Compresor/descompresor	Sí	
cmake	Compilador	No	
openjpeg	JP2000 codec	Sí	2.12

Tabla 9: Aplicaciones del entorno de desarrollo

Es importante destacar que también han sido necesarios descargar e instalar librerías específicas para que las aplicaciones listadas en la tabla anterior funcionen correctamente.

NOMBRE DE LIBRERÍA	APLICACIÓN
build-essential	Qt, OpenCV
mesa-common-dev	Qt
libgl1-mesa-dev	Qt
libssl-dev	MySql
libmysqlclient-dev	MySql

libjpeg62-dev	ImageMagick
libgtk2.0-dev	ImageMagick
libavcodec-dev	OpenCV
libavformat-dev	OpenCV
libswscale-dev	OpenCV
libv4l-dev	OpenCV

Tabla 10: Librerías necesarias para la instalación de las aplicaciones de la Tabla 9

3.2.1.- Qt

La implementación del servidor web, del demonio, del proceso de descarga y del proceso de generación de línea de costa han sido implementados utilizando Qt. Qt que es una librería/biblioteca de clases para el desarrollo de interfaces de usuario basados en C++[7].

Debido a que Qt está basado en C++, una aplicación que se desarrolla con Qt puede ser compilada y ejecutada en Windows, Mac OSX, Linux y varias ramas de Unix, el CTTC ha elegido el uso de este framework para del desarrollo e implementación del prototipo descrito en este proyecto final de carrera.

Además Qt incluye soporte para otras tecnologías como OpenGL, XML, bases de datos, Webkit, multimedia, *multithreading*, etc. Dispone además de enlaces o *bindings* para utilizar otros leguajes de programación como Python, Ruby o C#.

Qt proporciona además un entorno de programación o IDE propio: Qt Creator así como un programa para la creación gráfica de los GUIs Qt Designer y un sistema de ayuda propio.

3.2.2.- Qt Creator

Es un entorno de desarrollo integrado (IDE) para la creación de aplicaciones que utilicen el entorno de programación Qt [8]. Entre las características más importantes de Qt Creator se puede citar:

- Permite el desarrollo de una aplicación de una forma sencilla con el Project *Wizard* o reanudar el trabajo mediante un acceso rápido y sencillo a las sesiones y proyectos más recientes.
- Proporciona el editor integrado (*Qt Designer*) para el desarrollo de aplicaciones con interfaz de usuario basado en las widgets de Qt.
- Construye, ejecuta y empaqueta proyectos basados en Qt para las plataformas soportadas.
- Realiza *debug* usando el entorno grafico
- Accede de forma rápida y sencilla a la documentación con el sistema integrado de ayuda.

3.2.3.- Librería QtWepApp

La implementación del servidor se llevará acabo utilizando una librería de clases implementada para Qt denominada ***QtWepApp*** [9].

Entre los motivos para su utilización destacan:

- Es una librería para Qt escrita en C++ que permite la utilización de un solo lenguaje de programación (a excepción de la UI) en todo el sistema.

- Al utilizar C++ se pretende optimizar el uso de memoria y recursos porque es un lenguaje de nivel medio.

- Se tendrá un servidor web, un demonio y dos procesos compilados, corriendo directamente sobre el sistema operativo.

- Es una imposición del CTTC.

QtWepApp es una librería que permite implementar un servidor HTTP en C++ y se distribuye bajo la licencia LGPL. La librería está diseñada para trabajar en sistemas operativos soportados por Qt.

El servidor HTTP procesa peticiones entrantes en hilos concurrentes. Trabaja con protocolos IPv4 yIPv6, conexiones persistentes, HTTPS, cookies, y subida de ficheros, además una ventaja particular de la librería es que posee un bajo requerimiento de memoria, alrededor de los 2MB, que permiten que el servidor pueda ser implementado en sistemas embebido [9].

3.2.4.- MySql Server

La implementación del modelo de datos se realiza mediante la utilización de un servidor ***MySQL***, debido a que está considerado como el sistema de bases datos *open source* más extendido [10], porque está desarrollado en C++ y porque Qt implementa clases que permiten su integración en el entorno de desarrollo.

MySQL es un sistema gestor de bases de datos relacional, se basa en el leguaje de consultas SQL (*Structured Query Language*) desarrollado bajo licencia dual GPL/Licencia comercial por Oracle Corporation.

3.2.5.- ImageMagick

Las imágenes que la *“API Hub”* de Sentinel ofrece están en formato JP2000, formato extensamente difundido para la compresión de imágenes cartográficas [11].

Por otro lado, el algoritmo de extracción de línea de costa diseñado por el CTTC trabaja con imágenes JPG, por tanto es necesario implementar una etapa de conversión de formatos utilizando las herramientas que proporciona ***ImageMagick***.

ImageMagick es un conjunto de utilidades de código abierto para desplegar, manipular y convertir imágenes a través de líneas de comando. Está disponible para plataformas Linux, Mac OS y Windows [12].

3.2.6.- OpenCV

OpenCV (*Open Source Computer Vision*), es una librería de código abierto dirigida a la visión por computador. Dispone de interfaces para diferentes lenguajes de programación, como C ++, C, Python y Java y es compatible con Windows, Linux, Mac OS, iOS y Android.

OpenCV cuenta con más de 2500 funciones [13], que incluye en un amplio conjunto de algoritmos de visión por computador, donde se pueden encontrar funciones dirigidas a detección facial, identificación seguimiento de objetos, extracción de modelos 3D, fusión de imágenes, reconocimiento del paisaje, realidad aumentada entre muchos otros. Además utiliza ***CMake*** como gestor de compilación.

3.2.7.- Rabbit VCS

Rabbit VCS es un conjunto de herramientas gráficas escritas para proporcionar acceso simple y robusto para sistemas de control de versiones [14].

Un sistema de control de versiones VCS (*Version Control System*) registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se pueda recuperar versiones específicas en cualquier momento

El "*Centre Tecnològic Telecomunicacions Catalunya*" tiene implementado un servidor para el control de versiones implementado con *Redmine.*

Redmine es una herramienta para la gestión de proyectos que incluye un sistema de seguimiento de incidentes con seguimiento de errores y un repositorio de control de versiones entre otras varias herramientas.

3.2.8.- AngularJS

AngularJS es un *framework* JavaScript de desarrollo de aplicaciones web en el lado del cliente, ha sido desarrollado por Google y utiliza el patrón MVC (Modelo-Vista-Controlador) aunque sus creadores lo definen más bien como un MVW (*Model-View-Whatever)* [15].

AngularJS adapta y amplía el HTML tradicional para servir mejor contenido dinámico a través de un data *binding* bidireccional que permite la sincronización automática de modelos y vistas.

Un *binding* es una asociación entre dos cosas, por ejemplo un nombre de variable con su contenido; estas asociaciones pueden ser realizadas en las diferentes etapas de un programa.

El enlace de datos (*Data-binding*) en una aplicación que utiliza AngularJS es una sincronizan automática de la información entre los componentes de la vista y el modelo.

La forma en que AngularJS implementa el *data-binding* permite tratar el modelo como una fuente de datos verdadera para la aplicación. La vista es una proyección del modelo en todo momento. Cuando el modelo cambia la vista refleja los cambios y viceversa. Sincroniza constantemente con la vista con el modelo y el modelo con la vista.

3.3.- Implementación

3.3.1.- Aplicación Web

La interfaz de usuario (UI), se encuentra implementada sobre una aplicación de una sola página SPA (*Single Page Application*) escrita en JavaScript, HTML y CSS3. Se la ha implementado utilizando AngularJS versión 1 y no se ha necesitado un entorno de desarrollo integrado (IDE) específico, se ha hecho uso del editor de texto denominado "Sublime".

Uno de los principales objetivos de las aplicaciones web SPA es conseguir la disminución de los tiempos de espera o latencia entre vistas, proporcionando una mejor experiencia de usuario.

Todas las vistas de la interfaz de la aplicación están contenidas en la SPA, realizando una única carga inicial, y solicitando sólo datos a la API del sistema cada que el usuario lo necesite. Los datos recibidos son estructuras JSON.

En esencia una SPA es la interfaz de la aplicación web implementada casi íntegramente en el navegador, aunque como toda página web tenga una base importante de HTML y CSS.

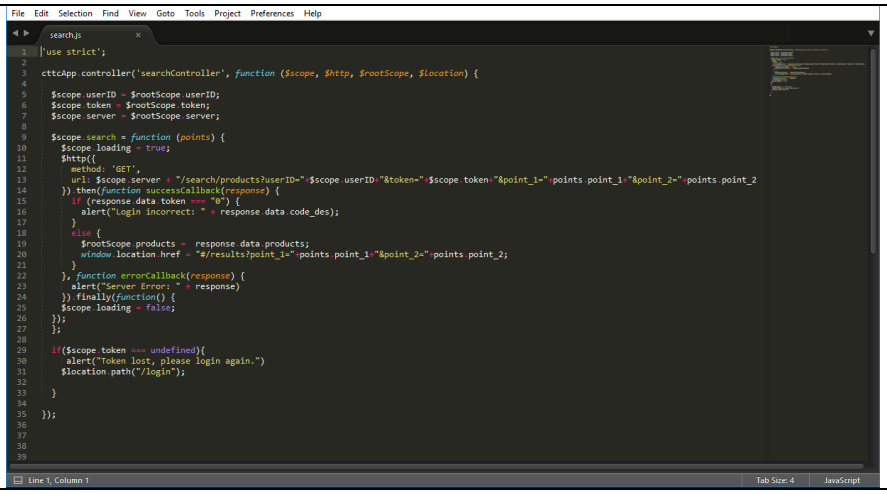


Figura 22: Vista del controlador "*search*" en el editor de texto Sublime

La aplicación web implementa un controlador para cada vista. Los controladores son los encargados de inicializar y modificar la información que contienen los *scopes* en función de las necesidades de la aplicación.

El objeto *scope* define la funcionabilidad de la aplicación, los métodos en los controladores, y las propiedades en las vistas. Los *scopes* sirven de nexo de unión entre el controlador y la vista.

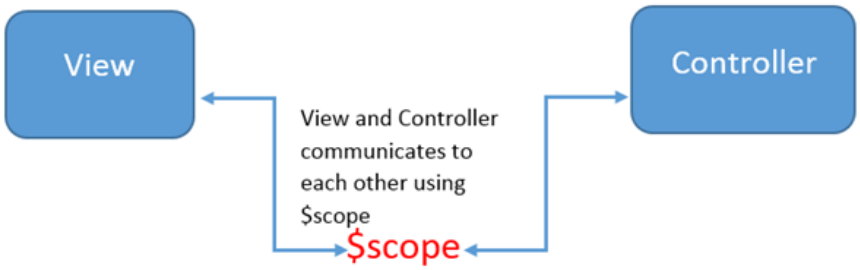


Figura 23: Los *scopes* sirven de nexo de unión entre el controlador y la vista

La siguiente tabla describe los controladores y vistas desarrollados:

CONTROLADOR	VISTA
Login.js	Login.html
Search.js	Search. html
Results.js	Results. html
Charts.js	Charts. html
Profile.js	Profile. html
About.js	About. html
Help.js	Help. html

Tabla 11:Controladores y vistas de la UI

```
<!doctype html>
<html>

<head>

<body ng-app="cttcApp" class='container'>
  <nav ng-hide="currentPath === '/'" class="navbar navbar-default">
  <div class="container-min-height" ng-view="">

  <div ng-hide="currentPath === '/'" class="footer panel-footer text-center">

  <script src="scripts/vendor/jquery/jquery.min.js"></script>
  <script src="scripts/vendor/bootstrap/bootstrap.min.js"></script>
  <!-- the angular core and the ngRoute module -->
  <script src="scripts/vendor/angular/angular.min.js"></script>
  <script src="scripts/vendor/angular-route/angular-route.min.js"></script>
  <!-- the main angular app code -->
  <script src="scripts/app.js"></script>
  <!-- Directives -->
  <script src="scripts/directives/header.js"></script>
  <!-- Controllers -->
  <script src="scripts/controllers/main.js"></script>
  <script src="scripts/controllers/about.js"></script>
  <script src="scripts/controllers/charts.js"></script>
  <script src="scripts/controllers/login.js"></script>
  <script src="scripts/controllers/search.js"></script>
  <script src="scripts/controllers/results.js"></script>
  <script src="scripts/controllers/help.js"></script>
  <script src="scripts/controllers/profile.js"></script>
</body>

</html>
```

Figura 24: implementación de los controladores en la SPA

3.3.2.- Servidor Web

El servidor web es un servicio que se inicia automáticamente al iniciar el sistema operativo y atiende peticiones HTTP a través de un puerto específico configurable, por defecto se utiliza el puerto TCP 2000.
En los siguientes apartados se describirá la implementación de cada una de las funcionalidades descritas en la sección 2.7.2.

3.3.2.1.- Mapeo de los recursos de la API

Como se ha visto en la sección 2.7.2, la aplicación web tiene funcionalidades que son alimentadas por datos que provienen de la API del sistema que se obtienen a partir de peticiones HTTP.

El mapeo de los recursos de la API se realiza gracias a la librería QtWebApp, cada recurso estará asociado a un controlador implementado a través de la clase ***HttpRequestHandler***, que permite generar respuestas una solicitud HTTP [16].

En la siguiente tabla se lista los controladores implementados con sus respectivos recursos.

DESCRIPCIÓN	RECURSO	CONTROLADOR	PARÁMETROS
Autentificación	/login	LoginController	Usuario Contraseña
Realizar búsqueda de imágenes	/products/search	SerachController	
Solicitar datos de usuario	/users/get	UserController	Token ID de usuario
Actualizar datos de usuario	/users/update	UpdateUserController	Token Datos de usuario en formulario
Crear tarea	/tasks/create	CreateTaskController	Token ID Usuario
Listar tareas de usuario	/tasks/get	GetTasksController	Token ID Usuario
Cancelar ejecución de tarea	/tasks/cancel	CancelTaskController	Token ID Usuario
Desautentificación	/logout	LogOutController	Token

Tabla 12: Controladores implementados en la Aplicación Web

Es importante destacar que los controladores de la aplicación web no necesariamente guardan relación con los controladores implementados en el servidor web. Los primeros responden al manejo de vistas utilizando AngularJS en un dominio JavaScript, que se ejecuta del lado del cliente sobre un navegador web, mientras que los segundos se ejecutan del lado del servidor y están implementados en Qt/C++.

Cuando el servidor web recibe una petición, un controlador que actúa como "*broker*" denominado ***RequestMaper***, encamina dicha petición HTTP en función de la URI del recurso solicitado.

En la figura se observa cómo una petición que solicita el recurso para listar tareas (/tasks/get) se encamina a través del controlador ***RequestMaper***, hacia el controlador ***GetTasksController***. Una vez que este controlador valida los parámetros recibidos (en el ejemplo los parámetros son correctos), genera una respuesta HTTP con datos estructurados en un JSON.

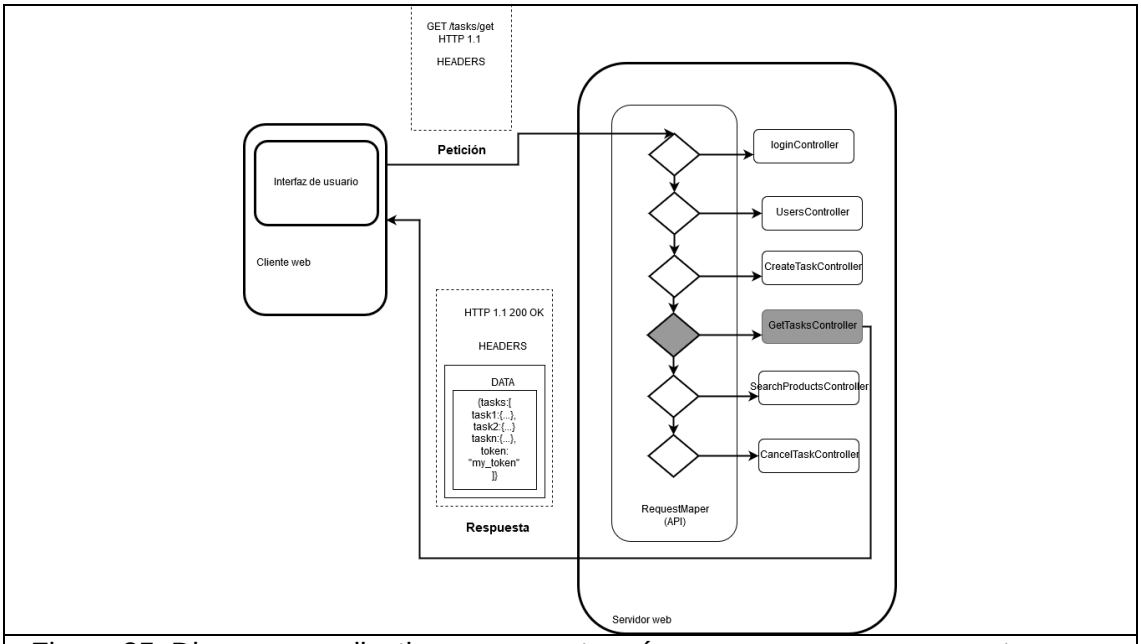


Figura 25: Diagrama explicativo que muestra cómo se genera una respuesta a una petición de un recurso para listar tareas (tasks/get)

```h
#ifndef REQUESTMAPPER_H
#define REQUESTMAPPER_H

#include "httprequesthandler.h"
#include "httpsessionstore.h"
#include "helloworldcontroller.h"
#include "listdatacontroller.h"
#include "logincontroller.h"
#include "userscontroller.h"
#include "orderscontroller.h"
#include "createtaskcontroller.h"
#include "searchproductscontroller.h"
#include "gettaskscontroller.h"

class RequestMapper : public HttpRequestHandler {
    Q_OBJECT
public:
    RequestMapper(QObject* parent=0);
    void service(HttpRequest& request, HttpResponse& response);
    static HttpSessionStore* sessionStore;
private:
    LoginController loginControllery;
    UsersController usersController;
    CreateTaskController createTaskController;
    GetTasksController getTasksController;
    SearchProductsController searchProductsController;


};

#endif // REQUESTMAPPER_H
```

```cpp
#include "requestmapper.h"

HttpSessionStore* RequestMapper::sessionStore=0;

RequestMapper::RequestMapper(QObject* parent)
    : HttpRequestHandler(parent) {
    // empty
}

void RequestMapper::service(HttpRequest& request, HttpResponse& response) {
    QByteArray path=request.getPath();
    qDebug("RequestMapper: path=%s",path.data());


    //Controlador para autentificación
    if (path=="/login")
    {
        loginControllery.service(request, response);
    }
    //Controlador de usuarios
    else if (path=="/users")
    {
        usersController.service(request, response);
    }
    //Controlador para listar tareas de usuario
    else if (path=="/tasks/get")
    {
        getTasksController.service(request, response);
    }
    //Controlador para crear tarea
    else if (path=="/tasks/create")
    {
        createTaskController.service(request, response);
    }
    //Controlador de búsqueda
    else if (path=="/products/search")
    {
        searchProductsController.service(request, response);
    }



    else {
        response.setStatus(404,"Not found");
        response.write("The URL is wrong, no such document.");
    }

    qDebug("RequestMapper: finished request");
}
```

Figura 25: Fichero .h y .cpp de la clase implementada para realizar el mapeo de recursos en el lado del servidor *RequestMaper*

3.3.2.2.- Controladores

Los controladores tienen por objeto atender peticiones de un recurso específico. Cada controlador atiende a un solo recurso. Cada controlador implementado dispone de cuatro funciones elementales:

1. Validación de token
2. Validación de parámetros recibidos
3. Procesamiento de la petición
4. Generación de respuesta

La validación del token se utiliza para autentificar al usuario. Un token es una cadena de caracteres generada a partir de una función *hash* de una clave general del sistema concatenado con el nombre de usuario y una función que depende de la hora de día.

Si el token no es correcto, el controlador generará una respuesta JSON del tipo 0, que contiene un mensaje de error. Por otro lado si el token es correcto el controlador validará los parámetros recibidos.

La validación de los parámetros recibidos es importante para evitar errores e incoherencias en las peticiones. Cada recurso tiene prestablecido que parámetros debe recibir y qué formatos deben tener. Si los parámetros no son correctos, también se enviará una respuesta JSON del tipo 0; si lo son, se procesará la petición.

El procesamiento que realiza cada controlador para atender un recurso es la característica que los diferencia. Por un lado tenemos controladores que generan un conjunto de datos que nutren a la aplicación y por otros controladores que simplemente confirman si una petición se realizó de forma correcta o incorrecta.

Dentro de los controladores que generan datos se encuentran:

- *LoginController*
- *SearchController*
- *GetTasksControlle*r
- *UserController*

El controlador de autentificación ***LoginController*** hace una consulta a la base de datos para verificar si el nombre de usuario y contraseña son correctos y después generar un token.
Por su parte, el controlador de búsqueda ***SearchController***, solicita un listado de imágenes de tipo Sentinel-2 al servidor Sentinel a partir de la latitud y longitud recibidas como parámetros. Posteriormente procesa la lista entregada por Sentinel en formato XML y finalmente genera una respuesta JSON con la lista de imágenes.

El controlador que lista las tareas que tiene un usuario ***GetTasksController***, consulta la base de datos a partir del ID de usuario recibido como parámetro y luego genera una respuesta JSON que contiene todas las tareas de un usuario.

UserController de forma análoga genera un JSON con los datos de usuario, a partir de una consulta a la base de datos utilizando el ID recibido de usuario. Que se ha recibido como parámetro

CreateTaskController, ***CancelTaskController y LogOutController*** atienden recursos que implican una acción, cuya respuesta sólo contiene información sobre si la petición se atendió de forma correcta o no.

3.3.3.- Demonio TCP

El demonio TCP tiene las siguientes funciones:

- Recepción de notificaciones TCP
- Gestionar procesos
- Envio de email.

3.3.3.1.- Recepción de notificaciones TCP

Para la recepción de las notificaciones TCP, el demonio utiliza las funcionalidades de servidor TCP/IP que proporciona una clase de Qt denominada ***QTcpServer*** [17]. Esta clase permite recibir conexiones TCP entrantes mediante un puerto especifico generando una señal ***newConection()*** cada vez que un cliente se conecta al a este servidor. La clase implementada se denomina ***TCPDaemon*** heredada de la clase ***QTcpServer.***

El servidor web, el proceso de descarga y de generación de línea de costa son los clientes que se conectarán al servidor TCP del demonio cuando envíen notificaciones de que ha sucedido un evento y su naturaleza es forma asíncrona. Por tanto el demonio, debe tener un servidor TCP capaz de atender varias conexiones a la vez.

Esta capacidad multihilo se realiza a través de *threads* que se crearán cada que se establezca una nueva conexión TCP válida, definimos como una conexión valida aquella conexión TCP que envíe, un mensaje, por ejemplo un carácter.

La creación de *threads* o hilos se ha implementado con la clase ***SocketThread*** heredada también gracias a una clase de Qt llamada ***QThread*** [18], que crea un nuevo socket TCP de tipo cliente para cada hilo.

En general la clase ***QTcpServer*** es un socket servidor, que cuando recibe una conexión entrante, arranca un *thread,* dicho *thread* crea un socket cliente que se conecta con el cliente que solicitó la conexión.

Una vez se ha recibido el mensaje TCP, el demonio llama a el método ***TaskProcess,*** método que gestiona todas las tareas y sus respectivos estados. Este método ha sido implementado específicamente para gestionar los procesos que atienden las tareas.

3.3.3.2.- Gestión de tareas

El método ***TaskProcess*** es el encargado de gestionar los procesos para llevar a cabo las tareas que el usuario ha generado.

Las tareas se encuentran almacenadas en la base de datos y para su gestión, se han definido 21 estados, agrupados en 7 grupos, cada grupo corresponde a una etapa de la tarea (ver Sección 2.7.2.4). Cada etapa tiene tres estados asignados (ver Tabla 6), y para pasar de un estado a otro, el estado previo debe haber sido ejecutado exitosamente.

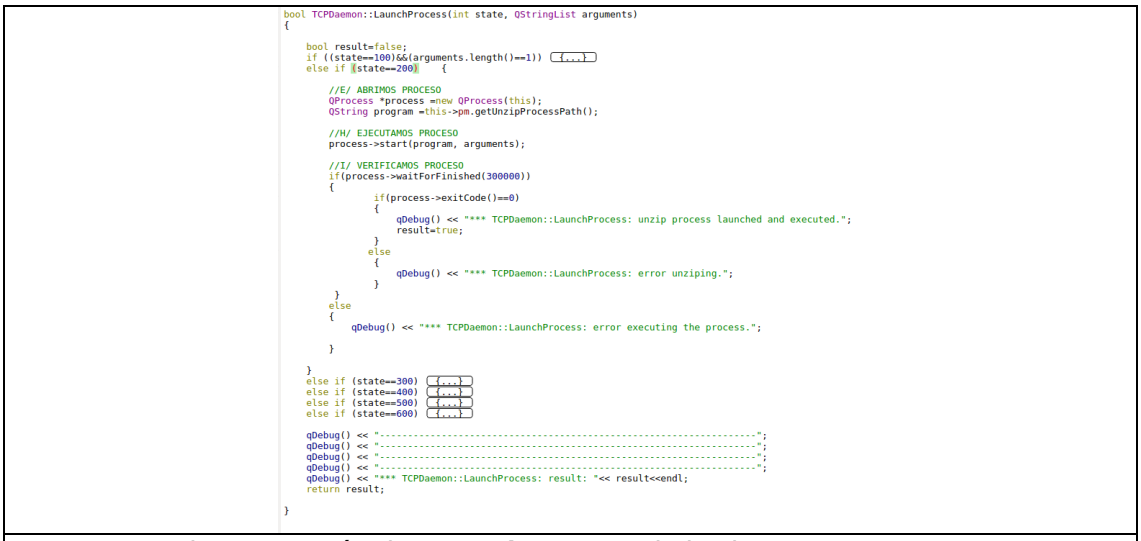
El método ***TaskProcess*** realiza consultas a la base de datos y confecciona una lista de tareas a ejecutar. De acuerdo al diagrama de clases expuesto en la Sección 2.7.5, cada tarea está relacionada con un producto y un usuario, además tiene un atributo que define su estado.

La lista de tareas posee un orden que depende de su estado y del atributo prioridad del usuario. Los estados del tipo x01 tienen prioridad sobre el estado x03 (ver Tabla 6) y el usuario tiene tres prioridades: “alta”, “media” y “baja”.

La lista de tareas está implementada en un *array* que contiene los identificadores de cada tarea ordenados en función de las prioridades. Las tareas que se encuentran en cola (x01) y que pertenezcan a un usuario con prioridad alta (3) se encontrarán en las primeras posiciones.

Una vez se ha generado el *array* de tareas, se lanzan los procesos en un orden descrito en la Tabla 1 de la Sección 2.5.5 mediante el uso de un método implementado denominado ***LaunchProces***.

LaunchProces utiliza una clase de Qt denominada ***QProcess*** [19] que permite iniciar programas externos ofreciendo métodos para comunicarse con ellos.



```
bool TCPDaemon::LaunchProcess(int state, QStringList arguments)
{

    bool result=false;
    if ((state==100)&&(arguments.length()==1)) {...}
    else if (state==200)    {

        //E/ ABRIMOS PROCESO
        QProcess *process =new QProcess(this);
        QString program =this->pm.getUnzipProcessPath();

        //H/ EJECUTAMOS PROCESO
        process->start(program, arguments);

        //I/ VERIFICAMOS PROCESO
        if(process->waitForFinished(300000))
        {
                if(process->exitCode()==0)
                {
                    qDebug() << "*** TCPDaemon::LaunchProcess: unzip process launched and executed.";
                    result=true;
                }
               else
                {
                    qDebug() << "*** TCPDaemon::LaunchProcess: error unziping.";
                }
         }
        else
        {
            qDebug() << "*** TCPDaemon::LaunchProcess: error executing the process.";

        }

    }
    else if (state==300) {...}
    else if (state==400) {...}
    else if (state==500) {...}
    else if (state==600) {...}

    qDebug() << "-----------------------------------------------------------------";
    qDebug() << "-----------------------------------------------------------------";
    qDebug() << "-----------------------------------------------------------------";
    qDebug() << "-----------------------------------------------------------------";
    qDebug() << "*** TCPDaemon::LaunchProcess: result: "<< result<<endl;
    return result;

}
```

Figura 26: Método ***LaunchPorcess*** de la clase ***TCPDaemon***

El control de los procesos lanzados se realiza por medio de la base de datos y de la comunicación que ***QProcess*** con el programa lanzado.

Los procesos de descarga y de generación de línea de costa, al ser diseñados en el presente proyecto, tienen la capacidad de actualizar su estado y progreso contra la base de datos, sin embargo, los procesos de descompresión/compresión y de conversión de formato, al ser diseñados por terceros no se comunican con la base de datos, por lo que la forma de tener conocimiento de su estado es a través del *exit code* que estos proporcionan cuando finalizan su ejecución y pueden ser comunicados a ***QProcess.***

Una vez se han atendido todas las tareas, el demonio se pone en modo "*standby*" a la espera de una nueva notificación TCP proveniente del servidor web (se ha creado una nueva tarea), el proceso de descarga (ha finalizado una descarga) o el generador de línea de costa (ha finalizado la generación de línea de costa).

3.3.3.3.- Envío de email

El tiempo empleado en realizar una tarea es variable, esto debido a la carga que presenta *Sentinel API Hub*, la cola de tareas existente en el sistema y a la velocidad de conexión entre el sistema y la API de Sentinel, y la carga de procesos que el ordenador se encuentre ejecutando. Es por ello que el sistema debe notificar al usuario cuando las tareas encomendadas hayan finalizado de forma exitosa o no.

Este tipo de notificación se ha implementado utilizando una librería SMTP de Qt. En este caso se ha visto conveniente no implementar un proceso independiente e implementar el envío de notificaciones por email implementando un método a la clase ***TCPDaemon***.

```
#ifndef TCPDAEMON_HPP
#define TCPDAEMON_HPP

#include <QTcpServer>
#include <SocketThread.hpp>
#include <iostream>
#include <QProcess>
#include <QVector>
#include <QtCore/qmath.h>
#include "dbinterface.h"
#include "processlauncher.h"
#include "preferencesmanager.h"
#include "imageextractor.h"
#include "smtp.h"

class TCPDaemon : public QTcpServer
{
  Q_OBJECT

  public:

     explicit TCPDaemon(QObject *parent = 0);
     bool startServer(int port_number);

 private:
     Task t;
     bool Download(DBInterface dbi, Task task);
     bool Unzip(DBInterface dbi, Task task);
     bool Extract(QString extension);
     bool JP2toJPG(DBInterface dbi, Task task);
     bool ImageProcess(DBInterface dbi, Task task);
     bool Zip(DBInterface dbi, Task task);
     bool Notification(DBInterface dbi, Task task);
     bool Extract(QString path_1, QString path_2);
     bool DeleteFileIfExists(QString path);
     bool DeleteDirectory(QString path);
     bool CreateDirectory(QString path, bool erase);
     bool UpdateTaskState(DBInterface dbi, Task task, int state, QString details);
     bool LaunchProcess(int option, QStringList arguments);
     int current_downloads;
     int current_unzips;
     int current_extractions;
     int current_conversions;
     PreferencesManager pm;


  public slots:

     void do_the_real_work(QString socket_data);

  protected:

      void incomingConnection(qintptr socketDescriptor);

};

#endif // TCPDAEMON_HPP
```

Figura 27: Cabecera de la clase ***TCPDaemon***

3.3.4.- Procesos

El sistema utiliza procesos desarrollados utilizando Qt/C++ y también utiliza procesos de terceros, la tabla siguiente indica qué procesos han sido diseñados e implementados y cuáles no, además en ella se observa los estados de cada proceso así como también los parámetros que necesitan para su ejecución.

NOMBRE	IMPLEMENTADOS POR TERCEROS	ESTADO INICIAL	ESTADO FINAL	PARÁMETROS DE ENTRADA
Proceso de descarga	No	100	103, 200	• ID de la tarea • Ruta destino de la descarga
Proceso de descompresión	Si (7Zip)	200	203,300	• Ruta de fichero a descomprimir
Proceso de conversión de formato	Si (ImageMagick)	400	403,500	• Ruta de carpeta origen JP2 • Ruta de carpeta destino JPG
Procesos Generador línea de costa	No	500	503,600	• Ruta de carpeta origen JPG • Ruta de carpeta destino
Proceso de compresión	Si (7Zip)	600	603, 700	• Ruta de carpeta origen • Ruta de carpeta destino

Tabla 13: Resumen de las características del proceso

Los procesos son programas ejecutables independientes que se lanzan de acuerdo a un orden y a la disponibilidad de recursos del sistema. Hay que recordar que para la generación de línea de costa es necesario pasar por varias etapas en donde en cada etapa actúa un proceso. Cuando este proceso finaliza exitosamente, se avanza a la siguiente etapa, así hasta llegar a la etapa final.

El orden en que se lanzan los procesos viene dado por las estepas definidas en la sección 2.4.

3.4.4.1.- Proceso de descarga

Está desarrollado a partir de la clase de Qt ***QNetworkAccessManager***, de su módulo de red que permite enviar peticiones HTTP y recibir respuestas***.***

La clase implementada se denomina ***Downloader*** y se caracteriza por tener como atributos un objeto del tipo ***QNetworkAccessManager*** y otro del tipo ***QNetworkReply.*** Además dispone de dos slots [21], uno que generan una señal cuando se ha finalizado la descarga y otro que genera una señal periódicamente y que almacena en disco duro incrementalmente la descarga y actualiza el progreso de la misma. Además dispone de un método principal llamado ***doDownload*** que inicia la descarga.

```
#ifndef DOWNLOADER_H
#define DOWNLOADER_H

#include <QCoreApplication>
#include <QObject>
#include <QNetworkAccessManager>
#include <QNetworkRequest>
#include <QNetworkReply>
#include <QUrl>
#include <QDateTime>
#include <QFile>
#include <QDebug>
#include <QProcess>
#include <dbinterface.h>
#include <preferencesmanager.h>


class Downloader : public QObject
{
    Q_OBJECT

private:
    PreferencesManager pm;
    DBInterface dbi;
    Task task;
    QNetworkAccessManager *manager;
    QNetworkReply *network_reply;


public:
    bool isActive;
    explicit Downloader(QObject *parent = 0);
    //Método que inicia la descarag
    bool doDownload(Task t);

signals:

public slots:
    //Slot para la finalización de descraga
    void replyFinished (QNetworkReply *reply);
    /*Slot utilizado para:
      1: actualizar el proceso de la descarga
      2: almacenar incrementalmente el fichero mientras se descraga
    */
    void updateDownloadProgress(qint64, qint64);

};

#endif // DOWNLOADER_H
```

Figura 28: Cabecera de la clase downloader

3.4.4.2.- Proceso de Descompresión

Se ha utilizado el programa 7-Zip para realizar el proceso de descompresión debido a que es un compresor/descompresor libre (licencia GNU LGPL) y que está disponible para plataformas Linux, Windows y Mac OS. Además 7-Zip permite ejecutarlo como proceso en *backgroud* y puede recibir parámetros para comprimir y descomprimir.

Su implementación se ha realizado utilizando la clase ***QProcess*** a través de la implementación del método ***LaunchProcess*** que como se ha comentado en la Sección 3.3.3.2.

El demonio lanza el proceso enviándole una serie de argumentos descritos en la Sección 2.7.4.1. Si los argumentos no son correctos o incoherentes, 7-Zip finaliza su ejecución con un *exit code* mayor a cero.

3.4.4.2.- Proceso conversor de formato de imágenes

Para convertir las imágenes JP2000 que se han obtenido tras la descompresión de la carpeta *zip* después de la descarga, se ha utilizado el programa ***ImgeMagick*** que es un conjunto de utilidades de código abierto para mostrar, manipular y convertir imágenes, capaz de leer y escribir más de 100 formatos[20]. ***ImageMagick*** es publicado bajo la Licencia Apache.

Como se hizo en el caso de la descompresión, el demonio lanza el proceso a través del uso de la clase ***LaunchProcess*** enviándole los argumentos descritos en la sección 2.7.4.1. El control de los estados lo realiza también el demonio a través de los métodos que la clase ***QProcess*** ofrece.

3.4.4.3.- Proceso generador de línea de costa

Para la generación de la línea de costa se ha implementado en Qt código desarrollado experimentalmente por el CTTC en C++.

El código utiliza librerías ***OpenCV*** y a partir de las imágenes de las bandas espectrales B03, B08 y B11, genera una línea de costa (ver Anexo 1).

El resultado que entrega este proceso es una imagen JPG que contiene la línea de costa sobrepuesta a la imagen B08.

El proceso recibe como parámetros de entrada la ruta de tres imágenes en formato JPG:

1. Imagen de la banda B03
2. Imagen de la banda B08
3. Imagen de la banda B11

Como en todos los procesos, si los parámetros no son correctos el sistema finaliza su ejecución actualizando el estado de la tarea en la base de datos con un “503”, caso contrario, la tarea se actualiza con un valor de estado de “501”.

La generación de la línea de costa se realiza a partir de la detección de los contornos de relación que existe entre una versión de la imagen B03 y B11, previa normalización y acondicionamiento de dichas imágenes.
La línea de costa generada por el algoritmo, se dibuja sobre la imagen B08.

El objetivo del presente proyecto final de carrera no contempla el estudio de algoritmos de procesado digital de la imagen, además el algoritmo usado e implementado tiene carácter experimental siendo su implementación simplemente demostrativa.

En la figura siguiente se muestran 4 imágenes, la primera es una porción de terreno, y las tercer y cuarto son ampliaciones de la primera imagen con la línea de costa generada y sobrepuesta.

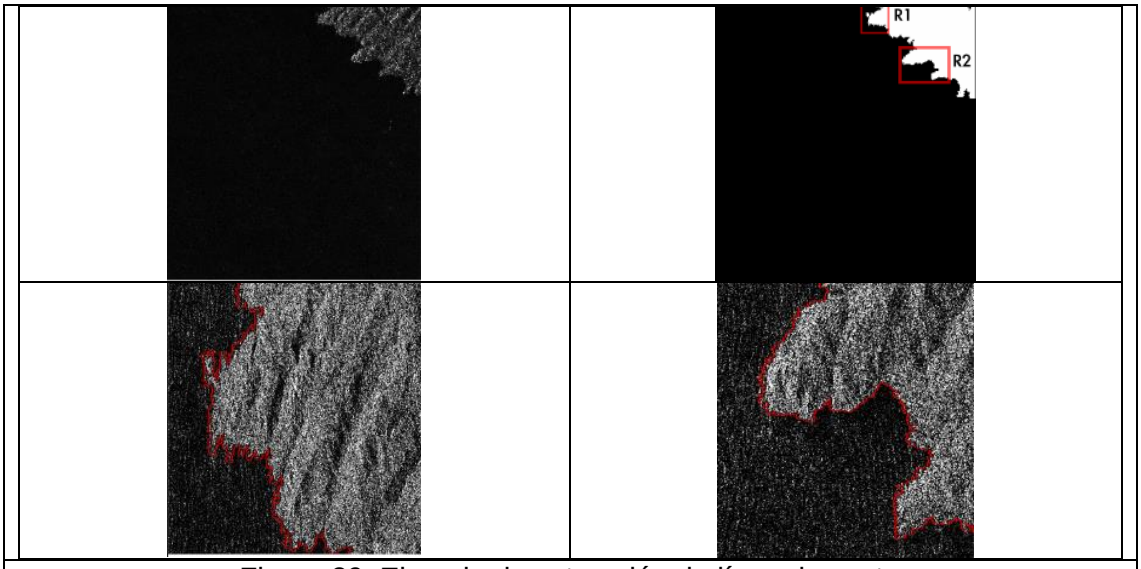


Figura 29: Ejemplo de extracción de línea de costa

3.4.4.4.- Proceso de Compresión

Es el último proceso, pero no la última tarea. El proceso de compresión comprime las N imágenes JPG que entrega el proceso de generación de línea de costa, en un fichero de formato zip.

De forma recíproca se ha utilizado el programa 7-Zip para realizar el proceso de compresión. En la sección 2.7.4.1 se señalan los parámetros que 7-Zip espera para realizar la compresión.

El control de los estados, se realiza de forma similar al caso de descompresión, a través del método ***LaunchProcess***, y del código de salida o *exit code* que 7-Zip genera cuando finaliza su ejecución. Cuando el procesos se lanza, el estado de la tarea se actualiza a “601”, una vez 7-Zip finalice su trabajo si el

código de salidas cero, el estado de la tarea se actualiza a “700”, si es mayor a cero “603”.

Una vez se haya comprimido el conjunto de imágenes con la línea de costa, el sistema, en particular el demonio enviará un correo electrónico con el enlace de descarga del comprimido al usuario que encargó la tarea de generación de línea de costa a través del uso de una clase de Qt para el envío de emails SMTP.

Además el usuario podrá acceder en todo momento a la interfaz de usuario para tener información sobre el estado de sus encargos.

CAPÍTULO 4. CONCLUSIONES

En la presente memoria se ha descrito el diseño y la implementación de un prototipo de sistema automatizado de producción de cartografía, a partir de imágenes satelitales Sentinel-2, con el fin de convertirse en un embrión de un sistema que a mediano-largo plazo, será implementado por el CTTC para cubrir diversas exigencias cartográficas, que, aparte de la generación de línea de costa, ofrecerá también un conjunto de aplicativos adicionales, para crear información cartográfica actualizada.

El objetivo principal de diseño e implementación de un prototipo se lo ha efectuado utilizando principalmente el framework Qt. El framework AngularJS ha sido utilizado sólo en la implementación de la interfaz gráfica.

La comunicación cliente-servidor ha sido posible gracias a la implementación de una API *RestFull* que a partir de recursos disponibles a través de URIs han permitido que la interfaz de usuario, implementada sobre una aplicación web, realice peticiones y despliegue información útil al usuario.

El modelo de datos ha sido el elemento sobre el que se apoya todo el sistema de gestión de procesos, el servidor web y por tanto la interfaz de usuario.

La generación de la imagen cartográfica ha servido para demostrar la utilidad del prototipo en la generación de información cartográfica actualizada a partir de las imágenes satelitales de Sentinel.

Cabe destacar, que la puesta en marcha del entorno de desarrollo ha requerido una dedicación adicional, debido, a que las aplicaciones utilizadas han tenido que ser compiladas específicamente para la distribución del sistema operativo, es el caso del driver SQL de Qt, ImageMagick y OpenCV.

Por último, destacar que existen varias futuras líneas de proyecto, entre las que destaca la normalización de cada etapa y su implementación modular, así el flujo de proceso podría ser programado por el usuario permitiendo reutilizar varios módulos o etapas, permitiendo implementar varios tipos de procesado sobre las imágenes Sentinel.

REFERENCIAS

1 "Sentinel-2"
[En línea]
https://sentinel.esa.int/web/sentinel/missions/sentinel-2
Septiembre de 2016

2 "Historia del Centre Tecnològic de Telecomunicacions de Catalunya"
[En línea]
http://www.cttc.es/about-cttc/history-cttc/
Septiembre de 2016

3 "About World Resources Institute"
[En línea]
http://www.wri.org/about
Octubre de 16

4 "Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)"
[En línea]
http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
Octubre de 16

5 "RFC 4287" - The Atom Syndication Format
[En línea]
https://tools.ietf.org/html/rfc4287
Octubre de 2016

6 "RFC 3986 - Uniform Resource Identifier (URI): Generic Syntax"
[En línea]
https://tools.ietf.org/html/rfc3986
Octubre de 2016

7 "Qt Documentation"
[En línea]
http://doc.qt.io/qt-5/index.html
Octubre de 2016

8 "QT Creator"
[En línea]
http://doc.qt.io/qtcreator/index.html
Octubre de 2016

9 "QtWebApp HTTP Webserver in C++"
[En línea]
http://stefanfrings.de/qtwebapp/index-en.html
Octubre de 2016

10 "Why MySQL?"
[En línea]
https://www.mysql.com/why-mysql/
Agosto de 2016

11 "Sentinel-2 MSI User Guide"
[En línea]
https://sentinel.esa.int/web/sentinel/user-guides/sentinel-2-msi/definitions
Agosto de 2016

12 "ImageMagick"
[En línea]
https://www.imagemagick.org/
Diciembre de 2016

13 OpebCV web page
[En línea]
http://opencv.org/
Diciembre de 2016

14 "RabbitVCS"
[En línea]
http://wiki.rabbitvcs.org/wiki/
Diciembre de 2016

15 "A quick look at Angular basics"
[En línea]
https://angular.io/docs/ts/latest/quickstart.html
Enero de 2017

16 "QtWebApp Documentation"
[En línea]
http://stefanfrings.de/qtwebapp/api/classstefanfrings_1_1HttpRequestHandler.html
Enero de 2017

17 "QTcpServer Class Documentation "
[En línea]
http://doc.qt.io/qt-5/qtcpserver.html
Noviembre de 2016

18 "QThread Class Documentation"
[En línea]
http://doc.qt.io/qt-4.8/qthread.html
Noviembre de 2016

19 "QProcess Class Documentation"
[En línea]
http://doc.qt.io/qt-5/qprocess.html
Diciembre de 2016

20 "ImageMagick - Supported Image Formats"
[En línea]
https://www.imagemagick.org/script/formats.php#supported
Diciembre de 2016

21 "How to Use Signals and Slots/es - Supported Image Formats"
[En línea]
https://wiki.qt.io/How_to_Use_Signals_and_Slots
Diciembre de 2016

ANEXOS

Anexo 1: Satélites Sentinel

Sentinel es una familia de misiones espaciales específicas del programa Copérnico; antes conocido como GMES (*Global Monitoring for Environment and Security*), "Vigilancia Mundial del Medio Ambiente y la Seguridad".

Copérnico es un programa de observación espacial, diseñado con el fin proporcionar información precisa, actualizada y de fácil acceso para mejorar y proporcionar herramientas para comprender y mitigar los efectos del cambio climático y garantizar la seguridad ciudadana. Esta iniciativa está liderada por la Comisión Europea (CE), en colaboración con la Agencia Espacial Europea (ESA).

Específicamente la misión Sentinel-2 está compuesta por un par de satélites que giran sobre una órbita polar con una separación de 180 grados entre un satélite y otro, además los límites de cobertura están acotados entre las latitudes 56° sur y 84° norte.

La misión monitoriza la variabilidad de las condiciones de la superficie terrestre, con un haz de cobertura y tiempo de revisita (10 días en el ecuador con un satélite y 5 utilizando 2 satélites) que permiten monitorizar por ejemplo cambios de vegetación entre estaciones y cambios entre el mar y tierra firme (líneas de costa).

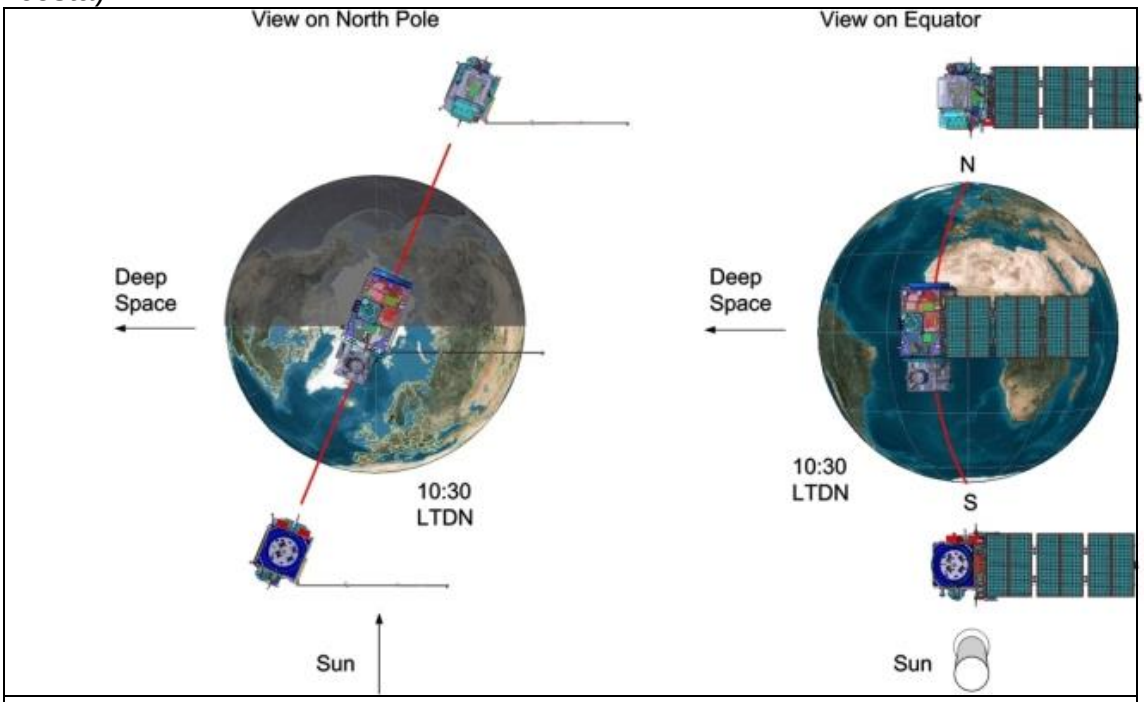


Figure A.1.1: The Twin-Satellite SENTINEL-2 Orbital Configuration

Sentinel-2 posee instrumentos que permiten obtener muestras en 13 bandas espectrales: cuatro bandas a 10 m, seis a 20 y tres a 60 m de resolución espacial. La muestra orbital se encuentra sobre los 290 Km.

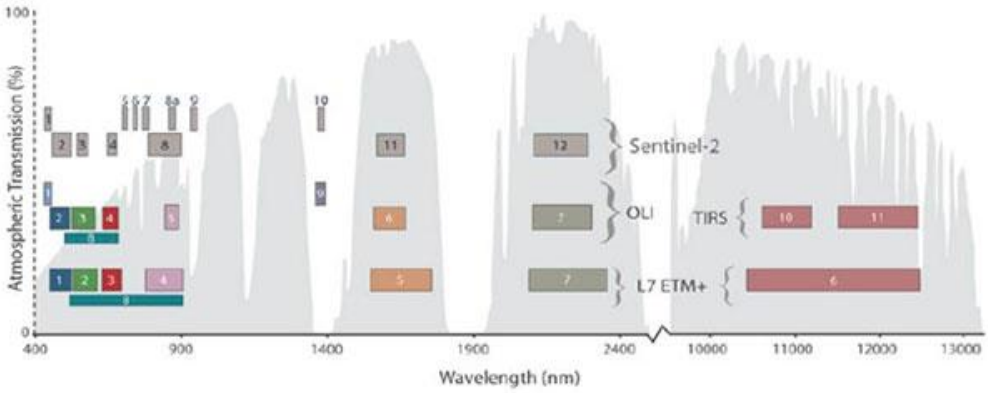


Figura A.1.2: Bandas SENTINEL-2 Fuente: USGS

Banda Sentinel-2	Nombre	Longitud de onda (µm)	Resolución (m)
Band 1 - Coastal aerosol	B01	0.443	60
Band 2 - Blue	B02	0.490	10
Band 3 - Green	B03	0.560	10
Band 4 - Red	B04	0.665	10
Band 5 - Vegetation Red Edge	B05	0.705	20
Band 6 - Vegetation Red Edge	B06	0.740	20
Band 7 - Vegetation Red Edge	B07	0.783	20
Band 8 - NIR	B08	0.842	10
Band 8A - Vegetation Red Edge	B8A	0.865	20
Band 9 - Water vapour	B09	0.945	60
Band 10 - SWIR - Cirrus	B10	1.375	60
Band 11 - SWIR	B11	1.610	20
Band 12 - SWIR	B12	2.190	20

Figura A.1.3: Bandas SENTINEL-2 Fuente: USGS

Las imágenes Sentinel-2, están disponibles a los usuarios de forma gratuita a través de "*The Sentinels Scientific Data Hub*" que es un punto de acceso libre y gratuito para la descarga de los "productos" Sentinel-1 y Sentinel-2.

El "*Data Hub"* dispone de una atractiva interface de usuario *"Scientific Hub"* que proporciona herramientas de búsqueda y filtrado de fácil manejo así como

también de una API denominada "*API Hub*" que está destinada a usuarios que utilizan scripts para automatizar la descarga de los "productos" Sentinel.

En este contexto, "producto" es un conjunto de datos que contienen una compilación de gránulos elementales de tamaño fijo, alrededor de una órbita y un gránulo es la mínima e indivisible porción de un producto que contienen todas las bandas espectrales.

Name	High-level Description	Production & Distribution	Data Volume
Level-1C	Top-of-atmosphere reflectances in cartographic geometry	Systematic generation and on-line distribution	500 MB (each 100x100 km2)
Level-2A	Bottom-of-atmosphere reflectance in cartographic geometry (prototype product)	Generation on user side (using Sentinel-2 Toolbox)	600 MB (each 100x100 km2)

Tabla A.1.3: Sentinel-2 product types Fuente: USGS

La tabla anterior indica los tipos de productos disponibles para Sentinel-2, en ambos productos, los gránulos, también denominados *"tiles"* o "baldosas", son proyecciones tipo UTM/WGS84 de orto imágenes de 100 x 100 Km^2, además las "baldosas" tienen un tamaño aproximado de 500MB cada una.

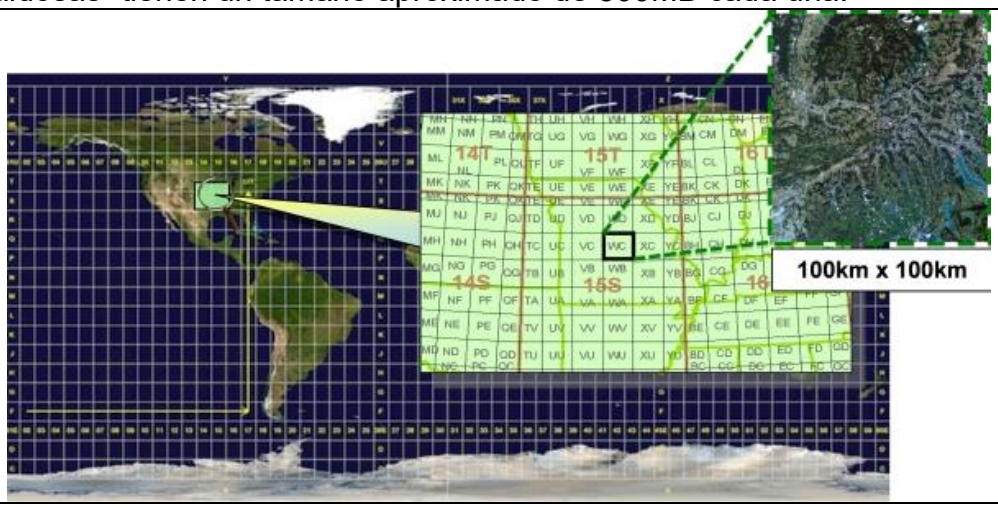


Figura A.1.4: Gránulos o porciones de terreno Fuente: USGS

Anexo 2: Descripción de casos de uso de la aplicación web

Las tablas siguientes describen los casos de uso de la aplicación web.

Nombre del Caso de Uso	Login	
Actor	usuario	
Descripción	El usuario introduce su nombre de usuario y contraseña	
Precondición	No hay precondición	
Flujo principal	Acción Actor	Acción Sistema
	El usuario introduce su nombre de usuario y contraseña	El sistema verifica las credenciales introducidas y despliega la pantalla de búsqueda
Flujo alternativo	El usuario introduce su nombre de usuario y contraseña de forma incorrecta	El sistema verifica las credenciales introducidas y despliega la pantalla de autentificación

Nombre del Caso de Uso	Búsqueda	
Actor	usuario	
Descripción	El usuario realiza la búsqueda de una imagen Sentinel-2 correspoendiente a una longitud y latitud	
Precondición	Estar autentificado	
Flujo principal	Acción Actor	Acción Sistema
	El usuario introduce latitud y longitud	El sistema busca imágenes relacionadas con la latitud y longitud introducidas y las despliega
Flujo alternativo	El usuario introduce latitud y longitud de forma incorrecta	El sistema lanza mensaje de error
Flujo alternativo	El usuario selecciona la pestaña de búsqueda "My Profile"	El sistema lanza la pantalla de "My Profile"
Flujo alternativo	El usuario selecciona la pestaña de búsqueda "My Charts"	El sistema lanza la pantalla de "My Charts"

Nombre del Caso de Uso	Crear tarea	
Actor	usuario	
Descripción	El usuario crea una tarea	
Precondición	Estar autentificado, búsqueda realizada	
Flujo principal	Acción Actor	Acción Sistema
	El usuario selecciona una imagen para procesar	El sistema crea una tarea relacionada con la imagen seleccionada
Flujo alternativo	El usuario selecciona la pestaña de búsqueda "search"	El sistema lanza la pantalla de búsqueda "search"
Flujo alternativo	El usuario selecciona la pestaña de búsqueda "My Profile"	El sistema lanza la pantalla de "My Profile"
Flujo alternativo	El usuario selecciona la pestaña de búsqueda "My Charts"	El sistema lanza la pantalla de "My Charts"

Nombre del Caso de Uso	Visualizar datos	
Actor	usuario	
Descripción	El usuario visualiza datos personales	
Precondición	Estar autentificado	
Flujo alternativo	Acción Actor	Acción Sistema
	El usuario presiona el bóton editar	El sistema el sistema activa el modo edición de las cajas de texto de entrada
Flujo alternativo	El usuario presiona el bóton cancelar	El sistema el sistema bloquea el modo edición de las cajas de texto de netrada
Flujo alternativo	El usuario presiona el bóton enviar	El sistema el sistema almacena las modificaciones hechas
Flujo alternativo	El usuario presiona el bóton enviar	El sistema el sistema despliega un mensaje de error
Flujo alternativo	El usuario selecciona la pestaña de búsqueda "search"	El sistema lanza la pantalla de búsqueda "search"
Flujo alternativo	El usuario selecciona la pestaña de búsqueda "My Profile"	El sistema lanza la pantalla de "My Profile"
Flujo alternativo	El usuario selecciona la pestaña de búsqueda "My Charts"	El sistema lanza la pantalla de "My Charts"

Nombre del Caso de Uso	Visualizar tareas	
Actor	usuario	
Descripción	El usuario visualiza tareas	
Precondición	Estar autentificado	
Flujo alternativo	Acción Actor	Acción Sistema
	El usuario presiona el bóton descargar resultado de tarea	El sistema el sistema inicia descraga local del resultado final de la tarea
Flujo alternativo	El usuario presiona el bóton cancelar	El sistema el sistema cancela ejecución tarea
Flujo alternativo	El usuario selecciona la pestaña de búsqueda "search"	El sistema lanza la pantalla de búsqueda "search"
Flujo alternativo	El usuario selecciona la pestaña de búsqueda "My Profile"	El sistema lanza la pantalla de "My Profile"
Flujo alternativo	El usuario selecciona la pestaña de búsqueda "My Charts"	El sistema lanza la pantalla de "My Charts"

Tabla A.2.1: Casos de uso de la interfaz de usuario

Anexo 3: Respuestas JSON de la API

El servidor web debe generar una serie de respuestas en función a las peticiones que, en este caso, la interfaz de usuario le realice. Las peticiones se han descrito en la Tabla 2 de la Sección 2.7.2.1, en donde por cada petición se lista el método, los *endpoints*, su función, los parámetros necesarios y el tipo de respuesta que la API del sistema genera.

Existen en general cinco tipos de respuestas que la API entrega que se lista a continuación, para más detalles favor de ver el Anexo 3.

TIPO DE RESPUESTA	DESCRIPCIÓN
Respuesta JSON 0	Respuesta a una petición que no se puede realizar.
Respuesta JSON 1	Respuesta a una petición realizada exitosamente
Respuesta JSON 2	Respuesta a una autentificación exitosa.
Respuesta JSON 3	Respuesta a una búsqueda exitosa.
Respuesta JSON 4	Respuesta a una solicitud de listar tareas.

Tabla A.3.1: descripción del tipo de respuesta que la API del sistema genera

```
{
"code":1001,
"code_des":"No valid input data",
"result":"false",
"token":"0"
}
```

Figura 3.A.1: Respuesta JSON 0

```
{
"code":2001,
"code_des":"Ok",
"result":"true",
"token":""bb22ee84c22b111aac3ef7f335070f3b"
}
```

Figura 3.A.2: Respuesta JSON 1

```
{
"code":2001,
"code_des":"Ok",
"result":"true",
"token":"bb22ee84c22b111aac3ef7f335070f3b",
"userID":"1",
"user_name":"leonardogsu"
}
```


Figura 3.A.3: Respuesta JSON 2


```
{
"products":[
{"date_1":"2016-03-19T21:10:15.247Z",
"date_2":"2016-03-19T09:04:49Z",
"date_3":"2016-03-19T09:04:49Z",
"details":"Date: 2016-03-19T09:04:49Z, Instrument: MSI, Mode: ,
Satellite: Sentinel-2, Size: 5.05 GB",
"length":"8544532822",
"name":"S2A_OPER_PRD_MSIL1C_PDMC_20160319T205829",
"s_id":"f238262b-6e46-4555-ab26-8de4aa840d2f",
"url_preview":"<url>/a123456789.jpg"
},
{"date_1":"2016-03-12T23:06:46.292Z",
"date_2":"2016-03-12T09:13:49Z",
"date_3":"2016-03-12T09:13:49Z",
"details":"Date: 2016-03-12T09:13:49Z, Instrument: MSI, Mode: ,
Satellite: Sentinel-2, Size: 5.16 GB",
"length":"8544532822",
"name":"S2A_OPER_PRD_MSIL1C_PDMC_20160312T225914",
"s_id":"6dda17e5-aa4e-4e19-b372-0e92257b4ee3",
"url_preview":"<url>/a123456789.jpg"
}
],
"token":"1234567890123456"
}
```


Figura 3.A.4: Respuesta JSON 3


```
{"tasks":
[{"details":"2017-01-22T00:19:26",
"id":40,
"priority_id":2,
"product":{
"date_1":"my_date1",
"date_2":"my_date2",
"date_3":"my_date3",
"details":"my_details",
"length":"my_length",
"name":"my_name",
"s_id":"my_s_id",
"url_preview":"url_preview"},
"status":"Error unziping",
"task_progress":100}],
"token":"bb22ee84c22b111aac3ef7f335070f3b","userID":1}
```


Figura 3.A.5: Respuesta JSON 4